Master of Science in Internetworking

Capstone Project Report

Work on ODL SDN controller to create PCEP / SR based tunnels and monitoring PCEP

Mostafa Safari

Supervisor

Prof. Ali Tizghadam

Winter 2022

Abstract

Segment Routing is a technology that is gaining traction to make the Multi-Protocol Label Switching network easier to manage. It has the benefit of being able to connect to a software-defined network. SDN is a well-established paradigm in the industry that makes network management simpler. Also, to boost network performance, SDN and SR are utilized.

Segment Routing is a source routing paradigm that addresses the MPLS network's shortcomings in simplicity and scalability. As the number of customers grows, scalability difficulties occur, limiting a network's performance. In the SDN era, segment routing was used to tackle these challenges and evolve a system.

In this project, we used OpenDaylight as an open-source SDN controller and its PCEP plugin to manage SR-TE tunnels in our network. Since other network controllers are also available, to hide this layer from higher system levels, we wrote a simple software that utilizes the ODL REST APIs to manage the SR-TE tunnels and provide simple APIs to this application’s user.

Table of Contents

1 Introduction

SDN (Software-Defined Networking) is a network architecture technique that allows networks to be intelligently and centrally controlled, or 'programmed,' using software applications. Regardless of the underlying network technology, this allows operators to manage the entire network consistently and holistically. [1]

A variety of competing factors surround enterprises, carriers, and service providers. Traditional business models are being devastated by the massive growth in multimedia content, the explosion of cloud computing, the impact of rising mobile usage, and continued commercial pressures to reduce cost while revenues remain static.

SDN allows software applications to program network behavior in a centrally controlled manner utilizing open APIs. Operators can manage the entire network and its devices consistently despite the complexity of the underlying network technology by opening up traditionally closed network platforms and creating a single SDN control layer. Services and applications running on SDN technology are abstracted from the underlying technologies and hardware that provide physical connectivity from network control. Instead of closely connected management interfaces, applications will interact with the network through APIs.

SDN promotes a vendor-neutral ecosystem while enabling multi-vendor interoperability. With open programmatic interfaces like OpenFlow, intelligent software can control hardware from different vendors. Intelligent network services and applications can also run in a common software environment from within the SDN. The ability for network operators to develop programs using SDN APIs and provide applications control over network behavior is a fundamental benefit of SDN technology. SDN enables users to create network-aware applications, monitor network conditions intelligently, and dynamically adjust network configuration as needed. [1]

There are many different SDN controllers available in the market, open-source SDN controllers like OpenDaylight and also controllers from vendors like Nokia and Cisco. For a large service provider like TELUS, it is not possible to rely on just one resource for its infrastructure. Also, it is necessary to test the solutions provided by vendors in practice and compare their performance with that of open-source controllers.

This project will be carried out as part of the TINAA program (TELUS Intelligent Network Analytics & Automation), and the main goal of this project is to be able to use different controllers so that the northbound does not notice any difference in connection to the different controllers, their use and their performance.

There are several items and topics that need to be learned prior to undertaking this project, including understanding the Segment-Routing and Path Computation Element Protocol, programming for OpenDaylight controller, understanding the structure of TINAA's entities and their interfaces.

The main objective of this project is to investigate SDN controllers' capability to create paths for traffic engineering and dynamically manage them in real-time. To that purpose, we'll use an open-source controller to create a proof-of-concept application that will dynamically establish and manage segment-routing oriented paths utilizing industry standard protocols. We also establish an abstraction layer with generic APIs exposed to northbound as part of the objective so that if the controller is altered, the northbound application is unaffected.

2 Segment Routing

This chapter delves deeper into the concepts and applications of Segment Routing. In addition, an examination of all protocols and technologies required for Segment Routing deployment may be found.

2.1 Segment Routing Introduction

Many MPLS networks today include Traffic Engineering (TE) features. A network operator can improve and make better use of its IP/MPLS network infrastructure via MPLS Traffic Engineering. Congestion avoidance is aided by TE, which also offers Fast Reroute (FRR) in the event of a link failure and allows a head-end router to re-optimize an existing TE tunnel path by utilizing newly available resources. All of this contributes to an MPLS label switched path's improved performance (LSP).

Currently, the go-to technology for network operators to build traffic engineered MPLS networks is RSVP-TE. But today, we are seeing the adoption of another protocol that not only helps enable traffic engineering but can also be used by software-defined networking (SDN) applications to automatically provision new paths.

MPLS-TE already exists as a traffic engineering solution. However, it has drawbacks in terms of scalability, manageability, and uses heavy signaling protocols such as RSVP-TE and LDP. Segment Routing overcomes these drawbacks and enables network service providers to change network behavior dynamically.

Segment Routing (SR) is a new source routing paradigm. It is a network technology that wants to address several drawbacks of existing IP/MPLS networks in terms of scalability, simplicity, and ease of operation [2]. Segment Routing is the basis of application engineered routing. Application engineered routing is a new business model that can enable applications to direct the behavior of the network. It is a paradigm designed and built for the SDN era.

Segment Routing is being standardized by Internet Engineering Task Force under Source Packet Routing in Networking (SPRING) group [2].

Segment Routing enhances packet-forwarding behavior. It allows the network to carry packets via a specific forwarding path. This path can be different from the natural shortest path that packet usually takes inside the network. Having the control to set up custom forwarding paths opens up many use case scenarios that certain applications can benefit from.

Source-based routing is not a brand-new idea in the world of networking, but it has not seen widespread adoption. A node (usually a router or a switch), which steers packets using a list of ordered instructions, is called a segment.

Today's traffic engineering solutions, such as Resource Reservation Protocol-Traffic Engineering (RSVP-TE), requires signaling for each path, and the state of each path needs to be present on each node that traffic traverses. Segment Routing can implement all these without the need for signaling protocol, making its architecture simpler and more scalable.

Segment Routing, using MPLS data plane, does not require Label Distribution Protocol (LDP) or RSVP-TE. Labels are distributed using Interior Gateway Protocol, either Intermediate System-to-Intermediate System (ISIS) or Open Shortest Path First (OSPF) and BGP. Running fewer protocols inside the network already makes the network more stable and scalable. Segment Routing paths are protected with Fast Reroute (FRR) capability, which allows rerouting of traffic in under 50 milliseconds in case of link or node failure.

Traditionally routers guide traffic inside the network primarily based on destination IP. Underlying Interior Gateway Protocol (IGP) was used to distribute network topology and compute the shortest path from ingress to the egress node. However, nowadays, packet loss, jitter, delay, and available bandwidth have become a major business differentiators when creating service-level agreements (SLAs). Therefore, these new business requirements are pushing networks to evolve towards more agility and flexibility.

Multiprotocol Label Switching (MPLS) introduced tunneling mechanisms and traffic steering functions. These were the main reasons behind the success of the MPLS technology. MPLS introduced MPLS-based Virtual Private Networks (VPN).

However, Resource Reservation Protocol-Traffic Engineering (RSVP-TE) did not have the same popularity as MPLS VPN. One of the main reasons of this was having poor load balancing characteristics. Another reason was that it was not very scalable. The final reason was that

computation was distributed, and this was causing some unpredictable traffic patterns and not the optimal use of resources.

The target audience for Segment routing is mainly Internet Service Providers (ISP), content providers, over-the-top (OTT) providers, large enterprises, data centers, and others.

However, to achieve this, some tools and protocols need to be present and enabled inside the network. In particular: SDN controller and protocols such as BGP, BGP-LS, IGP (IS-IS), PCEP, MPLS with Segment Routing.

SDN controller is needed to have a global view of the network communicating messages and commands back and forth with network devices. It acts as a medium between high-level applications and network devices.

BGP-LS and IGP protocols are needed to extract link-state information from the network. This data includes link bandwidth, metric, delay, and more. This data is readily accessible by the SDN controller and, therefore, by high-level application.

Path Computation Element (PCEP) needs to be present in the network in order to calculate suitable paths and then push the path onto the network node using the SDN controller.

Segment Routing can enable traffic engineering in three possible ways:

- By manually creating Segment Routing Label Switched Paths and explicitly defining routes inside the network. This equivalent of MPLS-TE but without extra protocols
- By manually creating Segment Routing tunnels. The path is calculated by Path Computation Element (PCE) and later pushed by SDN Controller onto the network
- Dynamically create Segment Routing tunnels. The path is calculated by PCE using existing network information or SLA, such as delay, bandwidth, metric, etc.

Segment Routing (SR) is not a new technology, but only recently has it been embraced by all the major network equipment vendors. It is a packet-forwarding technology where the source node defines the path for traffic, which is then sent through specific nodes and forwarding paths called segments. An SR path is not dependent on hop-by-hop signaling, Label Distribution Protocol (LDP) or RSVP. Instead, it uses segments for forwarding.

2.2 Segment Routing Concepts

This section discusses the main Segment Routing concepts. Firstly, a concept of a segment will be explained. After, the classification of segments will be represented with related examples.

2.2.1 Segment

According to the IETF, a segment is an instruction that the node executes on the incoming packet. This instruction could be, for instance, forward the packet to a specific network node according to the shortest path, or forward packet through a specific interface or deliver the packet to a given application or service.

A segment is identified with Segment Identifier (SID), and in the MPLS environment, it is encoded in 32 bits MPLS label [3].

2.2.2 Segment Advertising

Segments are advertised using IGP and BGP routing protocols. For both protocol types, Segment Routing extensions are defined to include Segment Routing information. In other words, routing protocols enable segments' signaling through the network. Let us now consider an autonomous system consisting of multiple IGP areas. Within each IGP area either IS-IS or OSPF is running. They are responsible for advertising segments within an IGP domain. However, in order to implement traffic engineering between an AS, segment exchanging between BGP peers must be enabled. BGP is extended to advertise the segments related to the BGP prefix.

Segment routing is constructed with SDN in mind. In software defined network it is assumed that SDN controller is in charge of determining end-to-end paths throughout a network. SDN has information on underlying network topology provided by BGP-LS protocol. In a software defined network that implements Segment Routing, BGP-LS is responsible to advertise SDN controller about segment identifiers. The topological path calculated by a SDN controller is pushed down to the source node in a form of the list of segments. Calculated path is carried by PCEP protocol. In SDN environment both PCEP and BGPLS extensions are necessary to support Segment Routing.

2.2.3 Global and Local Segments

According to its significance in the network all the segments can be divided on global and local. For now, the term network will be related to an IGP area. Global segment is related to the instruction that is supported by all nodes in an IGP domain. A global segment must be unique within a domain. Any node in an IGP domain must have all global segments in its Forwarding Information Base (FIB). The value of global segment identifiers is taken from the Segment Routing Global Block (SRGB). SRGB is a subspace of a 32bit SID space, and it takes values from 16000 up to 23999 [4].

Local segment is an instruction that is supported by the node originating it. Local segments take a value outside of SRGB range. Since it has only local significance, its value is related only to local router FIB. A router is not aware of local segments of the other routers in a domain. Moreover, the local SID values could be reused within an IGP domain, since a local SID value has local meaning for each single router.

2.2.4 IGP Segment Identifiers – IGP-SIDs

Link state protocols have an important role in Segment Routing. Global and local segments are distributed throughout the domain using IGP [3]. Both Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS) support Segment Routing thanks to well-defined protocol extensions. Segment Identifiers distributed by an IGP can be classified as it is shown in the following figure:

Figure 2-1 Types of SID

2.2.5 Prefix-SID

In general, Prefix-SID is a segment that refers to a specific network prefix. Prefix-SID is always global within an IGP domain and it refers to the shortest path computed by IGP to the related prefix. A packet that enters an IGP area with an active Prefix-SID will be forwarded along the

ECMP-aware shortest path to the prefix. Since a prefix could represent a node or a group of nodes within an IGP domain, Prefix-SIDs are further divided into Node-SIDs and Anycast-SIDs:

- Node-SID is a Prefix-SID and it refers to a specific node. A Node-SID has a global significance and it identifies exactly the prefix of the node's loopback interface. For instance, let's observe the following figure. 16004 is a Node-SID of the R4. The R1 wants to send a packet to R4 and it pushes the Node-SID on top of the packet's header. Since the shortest path to the R4 is through R2 and R3, R4 forwards packet to R2. When the packet arrives to R2, the router checks its FIB and passes the packet towards R3. Since R3 is not the destination of the label 16004, R2 does not remove the label. Since R3 is the last router towards destination it passes the packet to R4 and remove the label out of stack.

- Anycast-SID identifies a set of routers. A packet with Anycast-SID will be forwarded towards the closest node of anycast set. The Anycast-SID is an interesting tool for traffic engineering because it makes it easy to express macro traffic-engineering policies.

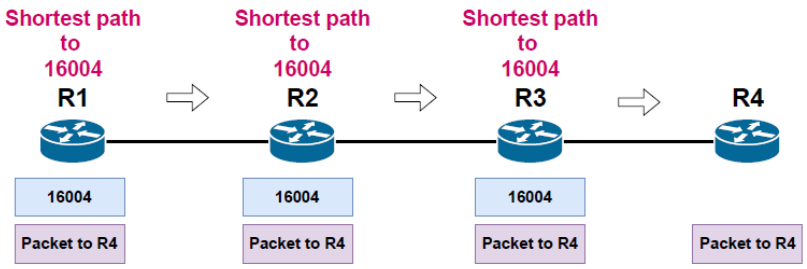


Figure 2-2

2.2.6 Adjacency-SID

The Adj-SID is IGP-SID that points on a specific link that belongs to the same IGP domain. Adj-SID has local significance, which means that a router maintains Adj-SIDs only for its neighbors. Adjacency segments must take a value that is outside of SRGB range. Usually a router allocates them dynamically. Since Adjacency SID's has local significance they don't have to be unique in the SR domain. Adj-SID is very useful if u want to steer traffic flow through a specific interface. The following figure illustrates how it works. Let's observe the R4. Adj-SIDs are assigned automatically for its three interfaces 24001, 24002 and 24003 (note that values are out of SRGB).

If one wants to use a link between R4 and R5 it is enough just to push the local label (24002) and packet will be forwarded to the next hop.

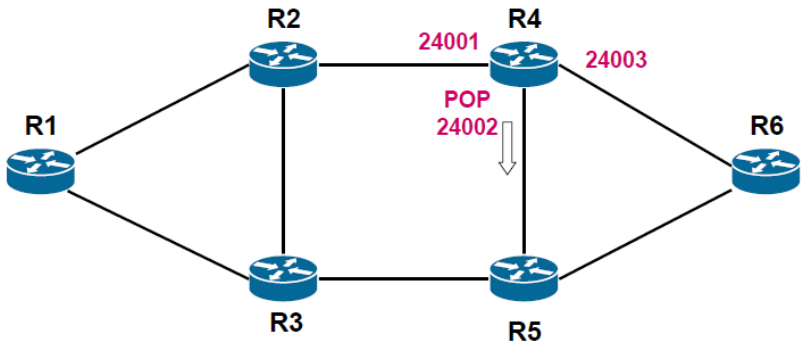


Figure 2-3

2.2.7 Routing Operations

Source node steers the incoming traffic flow by attaching an ordered list of SIDs to a packet header. The top segment is the first one that will be executed. Once the segment is executed (packet reaches an intermediate destination), next segment is going to be processed and so on. When last segment is executed, a flow either reaches its destination, or it just exits a SR domain and continues to be routed according to destination IP address. There are three actions that could be performed on segments by SR-capable nodes [5]. However, they are closely related to operations performed on

MPLS labels in MPLS networks. Segment Routing operations are:

1. PUSH (MPLS PUSH) – a segment is pushed on the top of segment stack

2. NEXT (MPLS POP) – an active segment is completed, and it is removed from the stack

3. CONTINUE (MPLS SWAP) – active segment is not completed yet and it remains active. Naturally, this operation exists only for global segments, since their execution could include multi-hops. Local segments (adjacencies) are executed in a single hop

The below figure explains how a packet is forwarded through a SR domain.

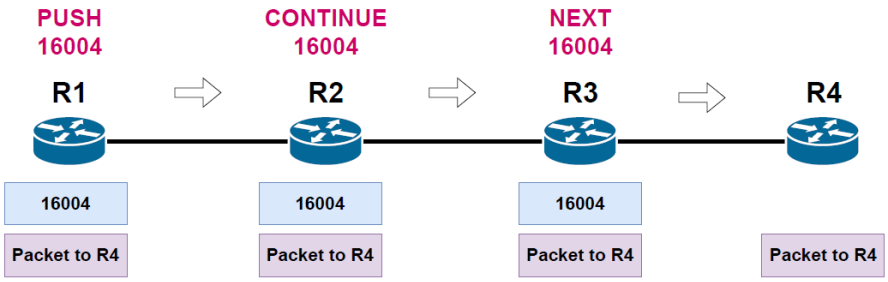


Figure 2-4 PUSH, CONTINUE, NEXT

2.3 Segment Routing Use Cases

2.3.1 Simplified transport of MPLS services

Segment Routing can offer the same tunneling service as MPLS in simplified manner using just IS-IS or OSPF. Service provider can easily enable services like L3VPN, VPLS and VPWS by setting up a Node-SID per network edge and ECMP tunnels will be created automatically from any ingress to any egress edge . LDP and RSVP are no more required, and that leads to following benefits:

- Simpler operation – less signaling in the network meaning the gain in terms of bandwidth and operation complexity
- Scaling – only one label for each node, reducing number of LSDB entries

2.3.2 Segment Routing and LDP coexistence

Inside MPLS Architecture, Segment Routing can coexist with LDP and RSVPTE [6]. Segment Routing Global Block assures that labels used for Segment Routing and LDP are allocated from different blocks of label. If both Segment Routing and LDP are enabled on the same router, LDP is given priority by default, but this can be changed using CLI configuration.

2.3.3 Traffic Engineering

Segment Routing can create tunnels according to customers' needs. SR enables traffic steering through any desired network path. By employing different SIDs, tunnels can be constructed in a smart way, which will result in increased network performance and throughput. Also, tunnels can

be designed by considering customers' SLAs. The most significant TE use cases are presented below.

Deterministic path or path avoidance is for sure the most useful tool in traffic engineering [7]. By exploiting adjacency SIDs, one can specify a path as path which flow will take through the network. Typical use case is presented on the below picture.

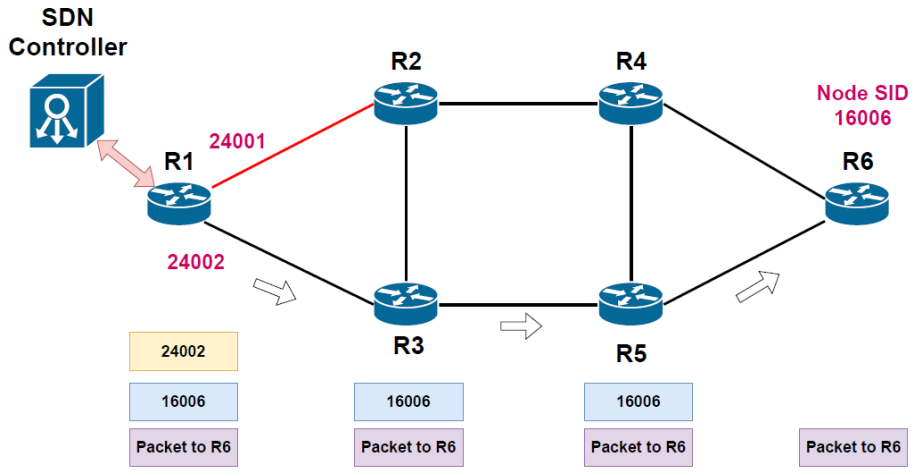


Figure 2-5 Segments determined by controller

R1 wants to send the traffic to R6 (Node-SID 16006). The easiest way to it is to push node segment on top of the packet and it will be forwarded according to the shortest path. However, Node-SID represents the instruction for ECMP-aware shortest path to R6, meaning that flow will take either R1-R2-R4-R6 or R1-R3-R5-R6. In the case the link R1-R2 becomes overloaded and the QoS drops down, a controller can dynamically push the traffic to R3 and avoid a busy link. Traffic will arrive at R3 and then it will continue to R6 according to the shortest path.

By assigning anycast SIDs, one can define a group of routers which flow will take on its way to the destination. For service providers this is very interesting tool because it can express macro policies such as "go via plane one of dual plane network" or "go via European Region" [8]. As an example, let's observe the following figure. The network can be described as a dual plane. One can steer the traffic only through yellow or blue nodes to the final destination by assigning labels {16001, 16005} or {16002, 16005}. ECMP is supported within a plain, meaning if there are disjoint paths, the load will be balanced (per flow).The main benefits are tunneling without RSVP

and LDP signaling, ECMP-aware routing and zero per-flow state on transient routers. Only additional anycast SID have to be configured (one per network plane).

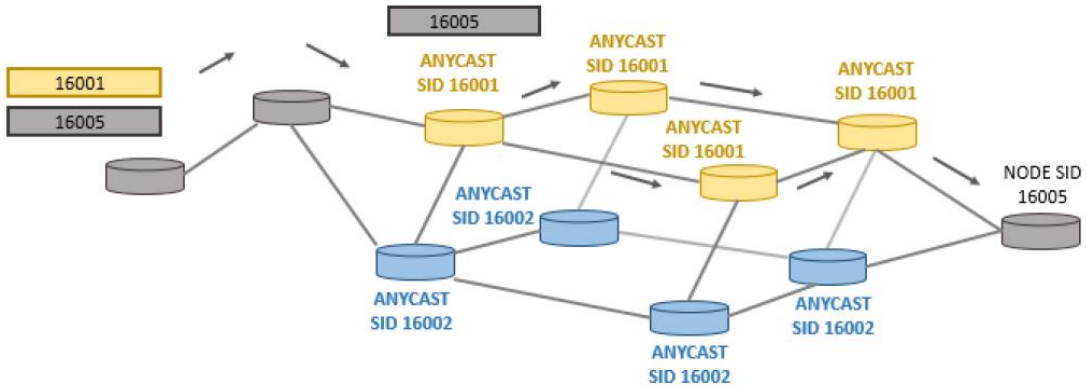


Figure 2-6

2.4 Multi-Protocol Label Switching

MPLS is a technology for data tunneling service build for high-performance telecommunication networks. MPLS operates on the so called 2.5 Layer and it is compatible with any network protocol of which IP is the most popular. MPLS has brought performance enhancements and new service creation capabilities in connectionless IP world. MPLS has introduced Virtual Private Network (VPN) services and QoS across the network [9].

In MPLS networks, packets are routed from one network node to the next based on 32-bit MPLS labels. In that way, a packet does not experience a delay caused by complex IP lookups in routing table, which can be especially crucial for high priority traffic such as voice, video and similar. MPLS tunnels are setup based on Forwarding Equivalence Criteria (FEC) [9]. When a tunnel is engineered by path calculation module, it is established using signaling protocols: Resource Reservation Protocol (RSVP) and/or Label Distribution Protocol (LDP) protocol. According to signaled information, each node on the route fills up MPLS routing table and reserve resources for a specific tunnel. Once the packet comes to ingress of MPLS area, it is processed by Label Edge Router (LER). According to the predefined policies, edge router attaches a designated label to the packet and passes it inside the MPLS network. When packet is received, transit or Label Switch Router (LSR), checks the MPLS label and according to the MPLS table, swaps the old label with

a new one, and passes the packet to the next router. When it comes to the end of the tunnel, egress LER, last label is removed and packet continuous with IP routing towards the final destination.

2.5 Segment Routing vs. MPLS

The following table presents the short comparison between Segment Routing and MPLS [10].

TECHNOLOGY FEATURES	SEGMENT ROUTING	MPLS
LABEL SIGNALING	IGP	LDP + RSVP-TE
IGP/LDP SYNC	NOT REQUIRED	REQUIRED
FAST REROUTE (FRR) 50ms	IGP	IGP+RSVP-TE
EXTRA STATES FOR FRR	NO	YES
OPTIMUM BACKUP	YES	NO
ECMP CAPABILITY	INBUILT	NO
TE STATE	ONLY HEAD-END	N^2 PROBLEM AT CORE NODE
SDN SUPPORT	YES	NO
ROUTING TYPE	SOURCE BASED	DESTINATION BASED
PATH CALCULATION	CONSTRAINED SPF OR PCE	IGP + RSVP-TE
SCALABILITY	HIGH	LOW
OPERATIONS & TROUBLESHOOTING	LOW	HIGH

Figure 2-7 Segment Routing vs MPLS

In MPLS network label signaling and resource reservation are done by implementing signaling protocols, LDP and RSVP-TE. As it was discussed before, in Segment Routing network it is enough to have an IGP protocol and once Segment Routing is configured, IGP will take labels and redistribute them within domain. There is no need to implement any other signaling protocol that is major benefit in terms of bandwidth and simplicity. Moreover, LDP has lot of drawbacks regarding synchronization after a link failure. In fact, after a failure LDP must synchronize with IGP that calculates the new set of shortest paths within the network. Since there is a time gap while LSPs become stable again, in some cases it may cause the loss of packets because core routers do not know how to forward packets that are addressed to external network. In Segment Routing only IGP is used and there is no need for synchronization with other protocols [11].

Having a good path protection is crucial for sensitive applications. In MPLS in some cases it is possible to have end-to-end path protection by calculating both primary and secondary path. For

both paths resources must be reserved using RSVP-TE protocol [12]. All routers that are included in primary and secondary path must maintain the state of tunnels. This guarantees QoS and no traffic loss in case of failure - the path is fully protected and reroute can be perform in very fast manner <50ms (FRR). However, double resource reservation is not efficient in terms of network resource utilization, especially in busy core networks. On the other hand, Segment Routing uses post-convergent path that is automatically calculated by IGP upon link failure and guarantees optimal path in new situation. There are no extra states that should be maintained to protect the path. The FRR mechanism in Segment Routing is called Topology Independent Loop-Free Alternate (TI-LFA) and it guarantees <50ms convergence [13].

Equal Cost Multipath enables traffic balancing among equal cost paths between source and destination [14]. In Segment Routing it is inbuilt - if there are two flows between the same source and destination (the same Prefix-SID) they will take different paths. This property supports network stability. In MPLS tunnels are determined strictly hop-by-hop meaning that ECMP is not supported.

In Segment Routing source routing paradigm is used, while in MPLS packets are tunneled by pushing the labels according their destination IP address. In Segment Routing network paths are determined by Constrained Shortest Path First (CSPF) algorithm. CSPF is extension of SPF algorithm. First, the shortest path algorithm is run, after which constrains are applied (available bandwidth, latency etc.) [15]. Path computation is usually done by an external entity such as SDN or PCEP. In MPLS paths are setup by combining IGP and RSVP-TE protocols.

Segment Routing was built for centralized data-plane network in mind. Even though, in theory, tunnels can be built manually, Segment Routing is fully supported by SDN paradigm. MPLS has a different technology approach -control plane is distributed and paths can be setup and maintained by utilizing distributed protocols. In such distributed environment it is very difficult to apply centralized control.

Segment Routing is highly scalable compared to MPLS. First of all, it eliminates need for signaling protocols which simplifies overall architecture and leads to simplified and cheaper hardware. Moreover, number of FIB entries is highly reduced by applying Segment Routing - each node should have approximately N-1+number_of_interfaces. In MPLS each intermediate router has

N^2 entries [16], which can create scalability problems in huge networks (e.g. Tier 1 core network).

At the end, Segment Routing simplifies overall operation and reduces need for network maintenance. Data plane is highly simplified since there are no signaling protocols. Furthermore, it enables easy operation by making labels constant over the network.

3 Path Computation Element Protocol

3.1 PCE Introduction

A Path Computation Element (PCE) is an element (most likely residing on a server) that specializes in complex path computation on behalf of its Path computation client (PCC). A PCE can be a router or a Server. In this project we would be focusing on PCE on a server.

Official definition of PCE is:

"A Path Computation Element (PCE) is an entity (component, application, or network node) that is capable of computing a network path or route based on a network graph and applying computational constraints." [17]

A Typical PCE Architecture consists of items that are introduced below.

3.1.1 Traffic Engineering Database (TED)

A PCE needs network resource information like topology, bandwidth, link costs, existing LSPs etc., which is stored in Traffic Engineering Database (TED). This information can be collected via peering with IGPs (OSPF, IS-IS) or BGP-LS which is the most used.

3.1.2 Path Computation Element:

PCE is responsible for doing the actual path computation based on the constraints provided and signaling that to the Path Computation Client (PCC). PCE specializes in complex path computation across various domains on behalf of its path computation client (PCC) with enhanced scalability.

3.1.3 Path Computation Client:

A Path Computation Client (PCC) is an element requesting PCE for path computation.

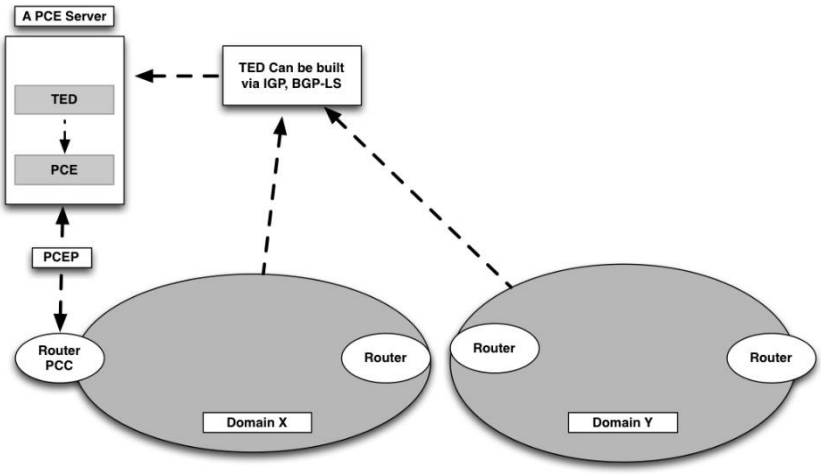


Figure 3-1 https://packetpushers.net/pce-pcep-overview/ Fig.8 [17]

3.2 Types of PCE

3.2.1 Stateless PCE

In the case of stateless PCE, it doesn't have knowledge of previously established LSPs. This severely limits a PCE capability to optimize the network resources.

Stateless PCE provides mechanisms to perform path computations in response to PCC requests. It utilizes only the Traffic Engineering database (TED DB) to do this computation.

3.2.2 Stateful PCE

In the case of stateful PCE, It keeps tracks of all the previously established LSPs (in LSP DB) and the available resources. Keeping a synchronized database with network state allows stateful PCE to make more optimal path computation decisions. So basically, it has LSPDB+TEDB+PCE in comparison to stateless PCE, which has only TEDB+PCE.

3.2.3 Passive Stateful PCE

In the case of a Passive Stateful PCE, PCC (router) is responsible for initiating path setup and retains the control on path updates. PCE receives the path request from the PCC, does the path computation and send it back to the PCC.

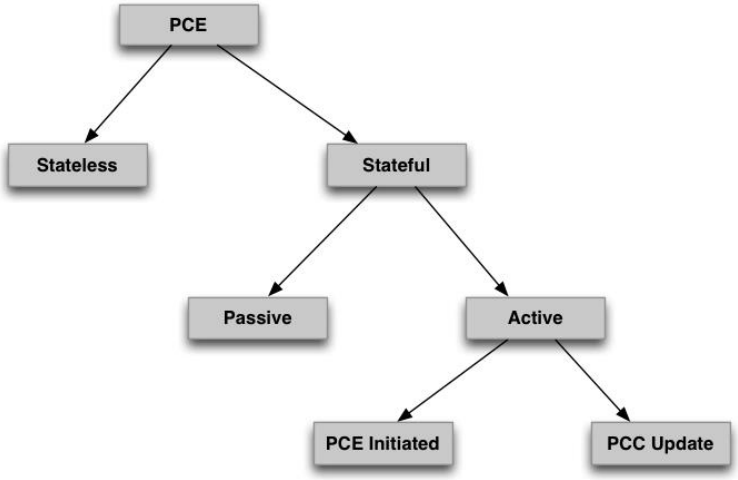


Figure 3-2Types of PCE [17]

3.2.4 Active Stateful PCE

In this case, a PCC allows the LSP to be delegated to PCE or a PCE can initiate an LSP as well. Basically, PCE can initiate LSP path setup and hence the term "Active" stateful PCE. Most of the work is being done in this area.

PCE Initiated: In this case an Active stateful PCE initiates an LSP and maintains the responsibility of updating the LSP.

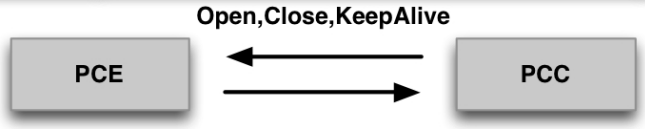
PCC Initiated: In this case a PCC initiates the LSP and may delegate the control later to the Active stateful PCE. A PCC can hand over the control to PCE and decided later to take it back. This back and forth switching happens with Delegated bit set during PCEP message exchanges.

3.3 PCEP

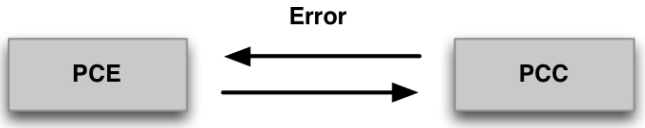
The first IETF draft describing PCEP was published in November 2005. The main protocol structure is specified in [18], and consists of a client-server interaction between PCC and PCE. Interaction is achieved through the exchange of PCEP messages running over TCP/IP, to exploit its reliability. Messages are defined to initiate, maintain and terminate a PCEP session. To perform path computations, PCC and PCE first open a PCEP session within a TCP session. The PCEP

session establishment includes the exchange of Open and Keepalive messages in order to agree on session parameters, such as timers and session refresh messages.

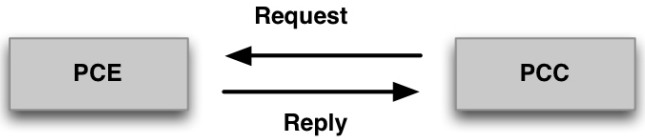
Messages	Purpose	PCE Type
Open, Close, Keep-alive	These are basically housekeeping messages. Open Message is also use for capability negotiation between PCE and PCC.	Stateless and Stateful



Messages	Purpose	PCE Type
Error Messages	Used for Error handling between PCE and PCC in case something goes wrong.	Stateless and Stateful



Messages	Purpose	PCE Type
Path Computation Request and Reply	PCE Request is sent by PCC to PCE asking for Path computation. PCE Reply is sent by PCE to PCC in response of PCE Request.	Stateless and Stateful



Messages	Purpose	PCE Type
PCC Report	PCC Report messages from PCC are used for multiple functions: 1) State Synchronization After the session between PCC and a stateful PCE is initialized, PCE must synchronize the state of a PCC‘s LSPs before it performs path computations or update LSP attributes in a PCC. 2) LSP State Report A PCC sends an LSP state report to a PCE whenever the state of an LSP changes. 3) LSP Control Delegation A PCC grants to a PCE the right to update LSP attributes on one or more LSPs; the PCE becomes the authoritative source of the LSP‘s attributes as long as the delegation is in effect; the PCC may withdraw the delegation or the PCE may give up the delegation.(active stateful PCE only)	Stateful PCE

Messages	Purpose	PCE Type
PCC Update	A PCE requests updates of attributes on a LSP. (active stateful PCE only). PCC reports back with the status report messages to the PCE.	Stateful PCE

Messages	Purpose	PCE Type
PCE Initiate	Used by PCE to Initiate LSPs on PCC. A PCC reports back on the status to the PCE.	Stateful PCE

3.4 Active Stateful PCE with SR

PCE is the brain for doing traffic engineering in the Segment Routing (SR-TE). It is responsible for doing the path computation and then sending appropriate label-stack (comprised of Node and Adjacency labels) to the Head end node. Then the headend node pushes those segments-list labels on the packets.

PCEP was also extended to support SR between PCE and PCC. Essentially few ERO Sub-objects were extended to support Node and Adjacency labels.

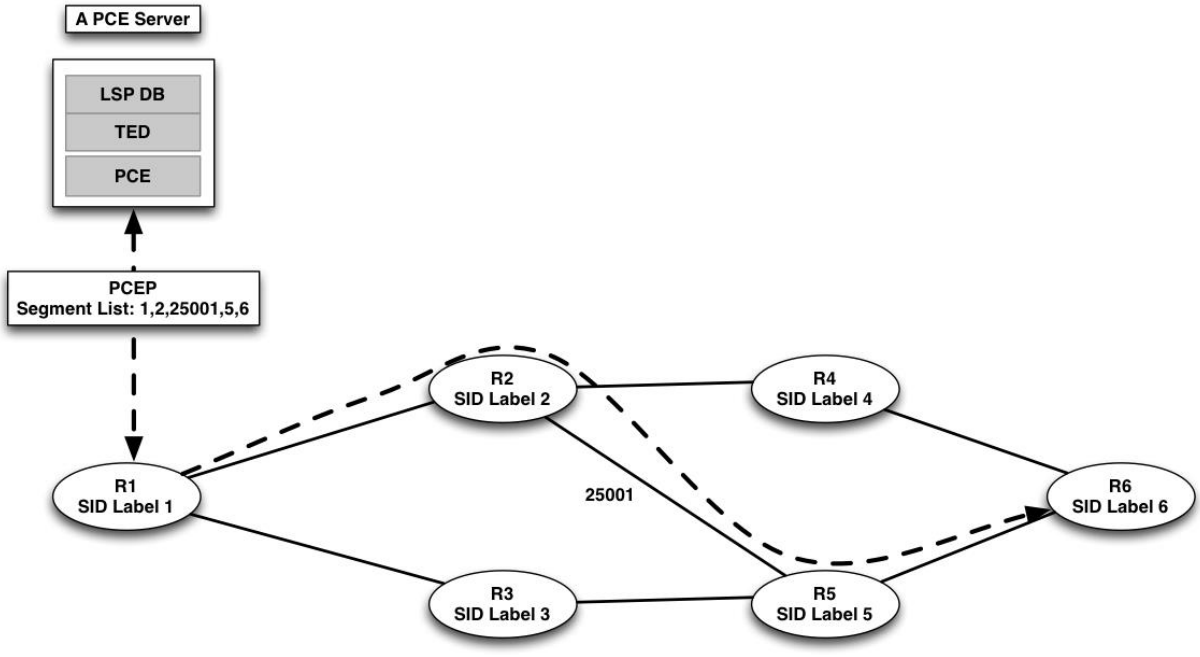


Figure 3-3Active Stateful PCE with SR [17]

4 Software-Defined Networking

4.1 Introduction

IP networks recorded the rapid growth in past decades, which made them more complex and consequently difficult to manage. The main limitation comes from the fact that today's networks are built of switches, routers and other devices that became complex because they must implement a number of distributed protocols and they use closed and proprietary interfaces. In this environment it is difficult and sometimes almost impossible for network operators, third parties and vendors to innovate [19].

Traditional IP networks consider the control and data plane tightly coupled and embedded in the same network node. In other words, control function is distributed over network devices meaning that each device is responsible to make a forwarding decision autonomously. In early stage of IP networks development this was considered a good aspect because it guaranteed network resilience. However, any change on decentralized control plane requires changes on all network devices manually. Lack of automation in network managing makes today's networks static and unable to adapt for real time demands [20].

To overcome such limitations, new networking paradigm has been proposed Software Defined Networking (SDN). In short, SDN can be defined as "an merging networking architecture where network control is decoupled and separated from forwarding mechanism and is directly programmable". In SDN architecture brings logically centralized control named SDN controller, which has a global view on underlying network. Low-level devices become strictly forwarding elements without any control function. All the instructions they receive from controller through specialized interface. The new protocols are defined for communication between controller and configurable switches. One of the most well-known protocols used by SDN controllers is OpenFlow. The main pillars of Software Defined Networks are:

- Separated data and control plane – control plane is removed from network devices and they became simplified forwarding elements
- Control functionality is placed on a dedicated entity called SDN controller

- Forwarding decisions are flow based. A flow could be defined as a packet stream between a source and destination that receive the same forwarding service.
- The network is programmable through software applications running on the top of controller

4.2 The SDN architecture

The SDN architecture generally has three functional groups [20], as it's depicted below:

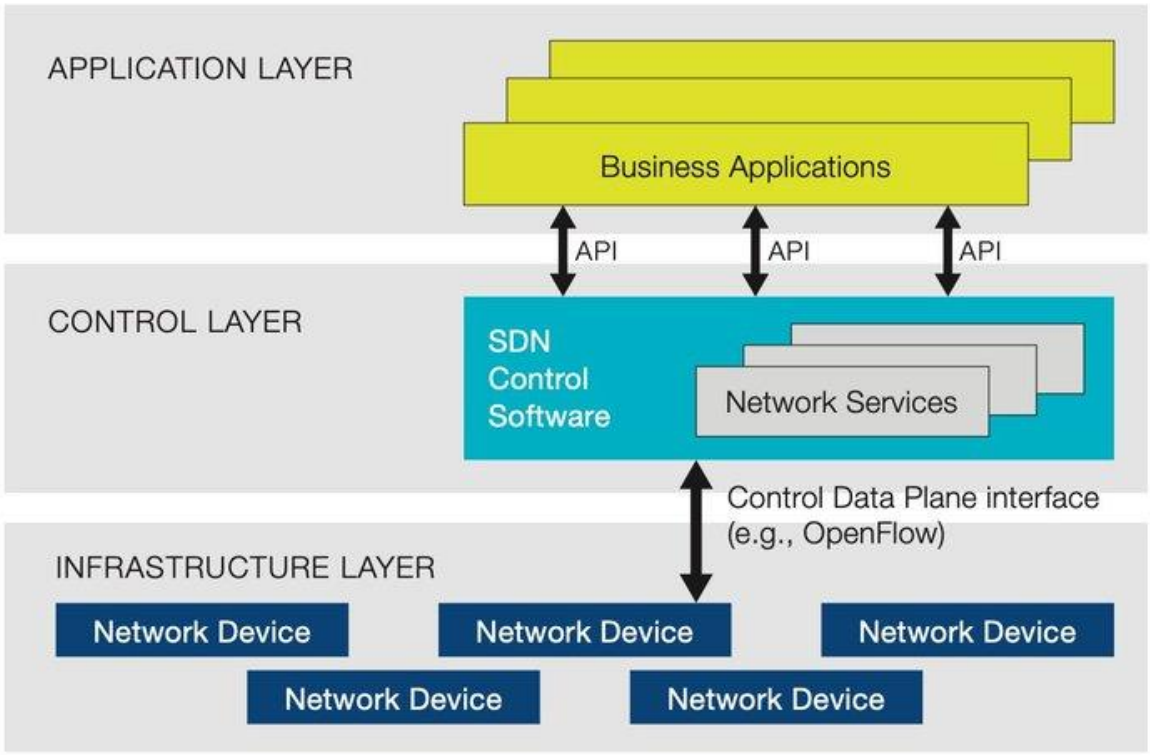


Figure 4-1SDN architecture [20]

Forwarding or data plane layer is placed on the bottom of SDN architecture. Data plane layer consists of switching devices connected in wired or wireless manner. Network devices perform set of elementary forwarding operations. They are programmable devices and they behave according to the instructions sent by controller.

The communication between SDN controller and programmable switches is enabled by southbound application program interfaces (southbound APIs). Southbound APIs facilitate

efficient network control and enable SDN controller to dynamically make changes in forwarding plane in real time. For example, SDN controller can add or remove an entry in forwarding table through southbound interface.

SDN controller is the “brain” of the network. It is centralized control point which manages flow control to the network devices below and the applications logic above. SDN controller makes abstraction view of the network, including statistics and the state of the network and sends it to the application level. Data plane could be controlled from the application level. Once instruction from the upper level is sent, controller takes it and forwards it to the lower level devices. The northbound API presents a network abstraction interface to the applications that sit on the top of SDN stack. This interface enables network programmability from application level. The northbound API is certainly the most critical part of SDN architecture. SDN controller is valued by innovative applications it can support, and northbound API must follow application requirements. Northbound APIs are used as well to connect SDN controller to automation stack and to orchestration platforms.

Application layer accommodate the set of applications that leverage functions offered by northbound API to implement operational logic and network control. From application level one can monitor physical network and control routing, firewalls, load balancers etc. All the commands coming from application layer are translated to southbound instructions that program behavior of forwarding devices.

4.3 Advantages of SDN

1. Better network control - SDN promotes a central point of control to distribute provider’s policies and configuration consistently throughout the network. SDN controllers provides complete visibility and control over network ensuring proper access control and traffic engineering
2. Orchestration of multi-vendor environments – SDN controller can configure and manage any SDN capable device. A single protocol is used for communication between a controller and devices of any vendor

3. Induces innovation – SDN gives possibilities to vendors, operators or a third party to develop applications, services and business models and trigger the revenue streams and more value from the network
4. Reduces operational expenditures - network hardware is simplified by removing control function. Overall operation costs are reduced by easier network control and better network utilization
5. Enhances network efficiency – centralized control and management increase automation and network orchestration. No need to configure individual network devices in forwarding plane to meet business policy changing. Network is directly programmable by a proprietary software or an open-source automation tool

4.4 Segment routing and SDN

Segment Routing was designed for SDN era. Segment Routing and SDN (SDN-SR) is very powerful combination and present a winning proposal for service providers. A SDN controller with a global view of the network it is capable to process business requirements and policies and translate them in Segment Routing paths. This leaves to service providers a huge number of possibilities to provide differentiated services and optimize their network.

SDN-SR is the perfect platform for application engineered routing. It gives possibility to an application to require specific path (in terms of latency, bandwidth, SLA) parameters and to push the packets through that specific path, without having to inform the network about it. That has reciprocal benefit for both application and network operation. Application can directly specify its requirements and push the traffic on optimal path. On the other hand, data layer is light-weighted because it doesn't have to maintain the traffic paths – they are directly specified from application [21].

In SDN-SR environment the network intelligence is combined [22]. Segments, as instructions, are designed in a smart and simple way to enable efficient traffic steering through the network. Segments give lot of possibilities to SDN controller how to express a wanted path. SDN controller intelligence is used to map the optimal path onto segments.

The key benefit of SDN-SR architecture is simplified control plane. Signaling protocols such as LDP and RSVP-TE are not necessary for SR functioning, which is direct benefit in terms of

simplicity and bandwidth relaxation. State is maintained only at the head-end router. Intermediate nodes do not have to maintain tunnel information that leads to improved scalability. Explicit routing is possible with or without ECMP – a controller can decide but stating proper SIDs. Automated FFR is guaranteed for any topology.

In reality, SDN controller should support the protocols that are essential for SR, PCEP and BGP-LS. SDN controller behaves as stateful PCE and can compute path in terms of segments and push it back to the PCC. As a property of stateful PCE, SDN controller can initiate PCEP session and perform flow optimization, if necessary. Topology information is obtained by configuring BGP-LS peering with BGP speakers. Each IGP domain must have at least one BGP speaker that will redistribute LSDB to SDN controller.

4.5 OpenDaylight

OpenDaylight project (OpenDaylight controller, ODL) is an open source SDN project governed by Linux Foundation [23]. Open source SDN controllers enable easy network testing and support network virtualization. Architecture of open-source solutions is typically modular meaning that controller consists of pluggable modules that perform different network functions. Open-source projects give possibility for development and customization. Today, there are many open-source projects launched for further development such as ONOS, OpenContrail, Pox, Ryu etc.

OpenDaylight project was announced back in 2013 with an aim to accelerate SDN development and industry adoption. ODL is based on Java programming language and supports OpenFlow standard [24]. Some of the companies that contribute ODL development are Cisco, Juniper Networks, VMware, Microsoft, Ericsson etc. Now fifteen releases are available:

Release Name	Release Date	Release Name	Release Date
Phosphorus (15)	September 2021	Oxygen (8)	March 2018
Silicon (14)	March 2021	Nitrogen (7)	September 2017
Aluminum (13)	September 2020	Carbon (6)	June 2017
Magnesium (12)	March 2020	Boron (5)	November 2016
Sodium (11)	September 2019	Beryllium (4)	February 2016
Neon (10)	March 2019	Lithium (3)	June 2015
Fluorine (9)	August 2018	Helium (2)	October 2014

4.5.1 ODL Architecture

Detailed OpenDaylight architecture diverse among releases [25]. Simplified ODL architecture presented below is common for all releases:

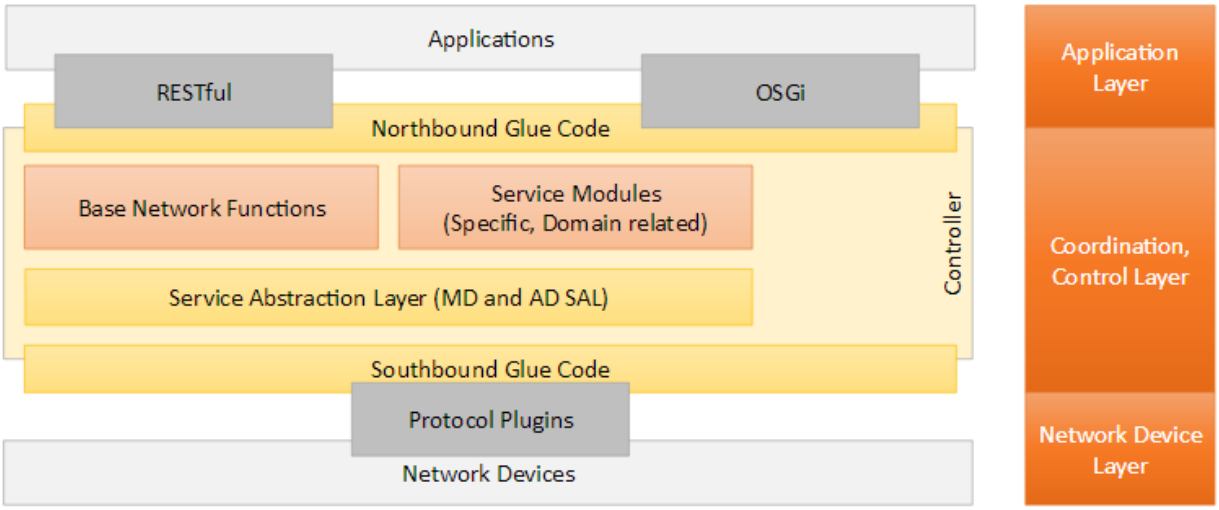


Figure 4-2Simplified ODL Architecture [25]

The controller acts like middleware in the OpenDaylight ecosystem. It is the framework that glues together the applications requiring services of the network devices and the protocols that talk to the network devices for extracting services. The controller allows the applications to be agnostic about the network device specifications, thereby allowing the application developers to concentrate on the development of application functionality rather than writing device-specific drivers.

As all SDN controllers, ODL consists of three main parts:

- Southbound APIs
- Control function layer
- Northbound APIs

Southbound Protocols

The southbound interface is capable of supporting multiple protocols (as separate plugins), e.g. OpenFlow, BGP-LS, LISP, SNMP, etc. These modules are dynamically linked to a service

abstraction layer (SAL), which determines how to fulfill the service requested (by applications) irrespective of the underlying protocol used between the controller and the network devices.

For example, an OpenFlow plugin will include the following: (a) connection, session, and state managers to manage the connection with the switches, (b) an error handing mechanism, (c) a packet handler to handle incoming packets from the switches, and (d) a set of basic services such as flow, stats, and topology.

Control Layer

The main components of ODL are service layer abstraction, service functions and pluggable modules. Service Layer Abstraction (SAL) represents a key bundle between service producers and consumers. Modules that provide services must register their APIs to the SAL registry. Whenever a request from service consumer comes, SAL binds them into 'contract'. There are two SAL architecture: application driven SAL and module driven SAL. As was mentioned before, an open-source project has pluggable module that enable particular function. However, there are some basic network functions that come as preconfigured part of controller. Some base network functions that come shipped with ODL are:

- Topology functions – a service for discovering network layout by subscribing to processes of network-link discovery
- Statistics services – for managing state of counters across the nodes, flows and queues
- Switch manager – stores discovered nodes
- Forwarding services – manage network flow state and forwarding rules

Platform services modules or vendor components enhance SDN controller functionality. Some of platform-oriented services are BGP-LS/PCEP that support traffic engineering, VTN (Virtual Tenant Network) component that enables network virtualization using OpenFlow, service function chaining that enables forming a ordered list of services, and etc.

Northbound Interface

ODL Controller exposes northbound APIs to the upper layer applications using OSGi framework or bidirectional REST APIs. REST APIs can be used by application that runs on the same computer as the controller or it can be totally different or remote machine.

REST is based on popular technologies such as HTML, JSON and XML that enables straightforward combining with programming language as Python, Java, C. Interaction with top level application is done through HTTP basic operations GET, POST, PUT and

DELETE. Data transmitted via REST API can be used on higher level to make higher-level business decisions, run algorithms, analytics etc. And results of this analytics can be channeled back to ODL Controller to for instance, create new rules in the network.

5 Implementation and Tests

5.1 Overview

The test environment in this project consists of a GNS3 model, which is part of a more extensive network, our Opendaylight controller, and a simple application above all others for managing SR-TE tunnels. In general, our environment is similar to the below diagram.

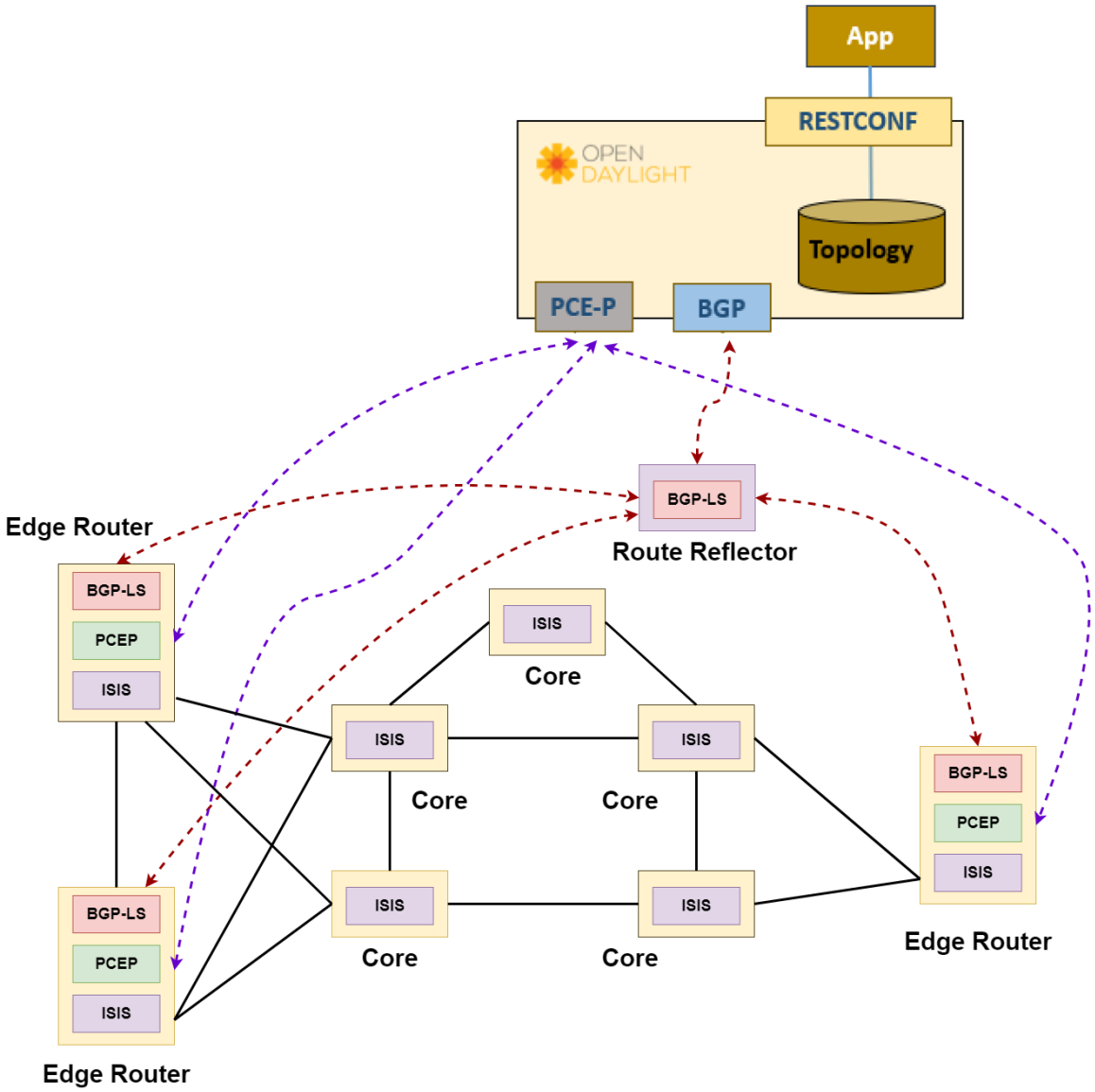


Figure 5-1 Network topology

In our GNS3 model, we have three edge routers: our PCC nodes. All routers are configured to use ISIS as the primary IGP routing protocol and support MPLS. Also, we configured all routers to enable segment routing, but only edge routers run PCEP, and core routers do not connect to the PCE, which is ODL in our case. Edge routers also use BGP-LS to redistribute the link-state information to the ODL. Edge routers are not directly connected to the ODL but through a route reflector.

Our application uses REST APIs to communicate with ODL and gets the link-state information and PCEP topology information. For now, this application can extract segments, links, and nodes information. Also, it can determine the available LSPs for each PCC node and their operational status in addition to their hops to the destination. We provided many functions which can be used to extract information and work with this system. Especially, we have two functions for creating and modifying the SR-TE tunnels, which we will discuss later.

5.2 GNS3 Model

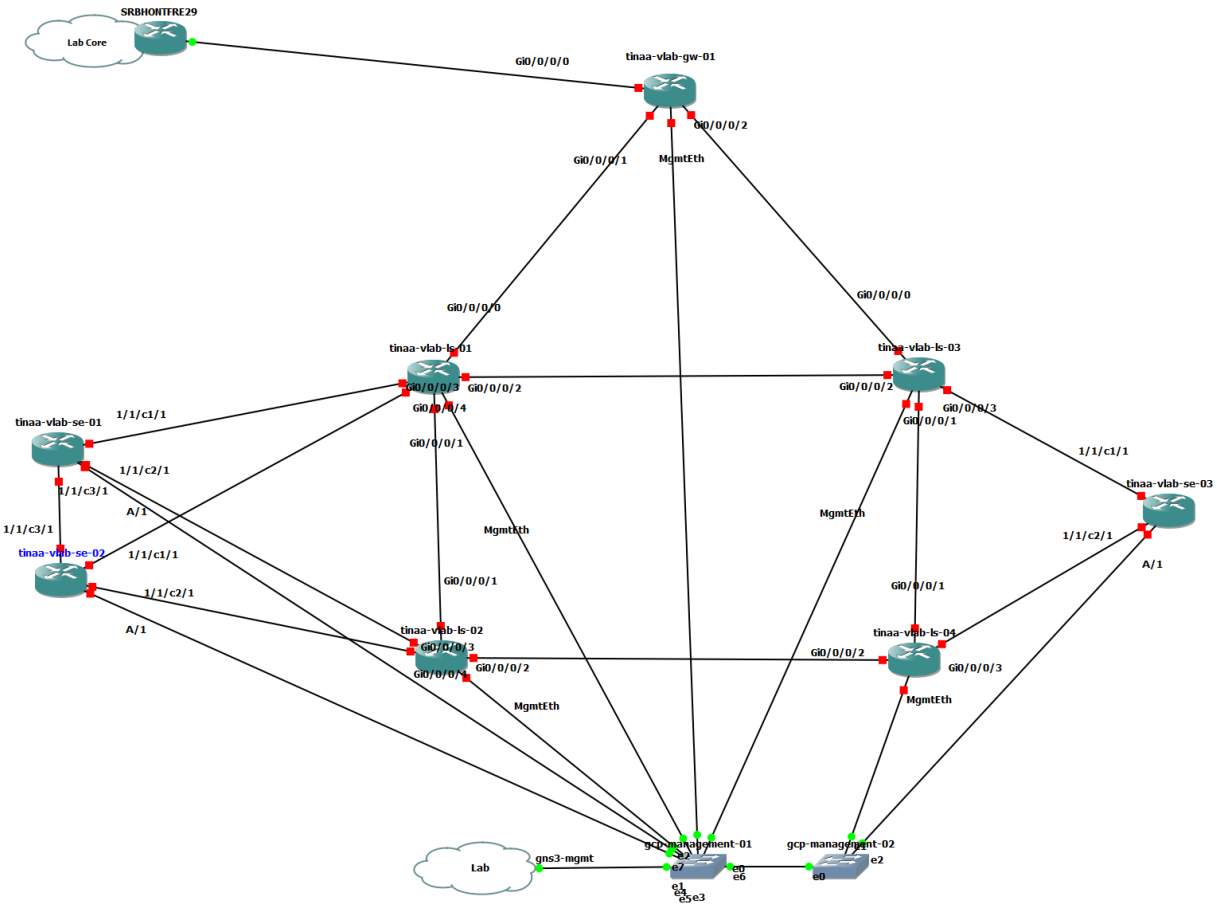


Figure 5-2GNS3 Network Model

The above figure shows the GNS3 model of our network. This is a section of a more extensive TINAA network, and we can see that the SRBHONTFRE29 router connects this model to the rest of the network through the cloud element in the GNS3. Management ports of the routers are connected to the switches and from there to the Lab network, where our ODL resides.

In our model, we have five core routers that only provide MPLS and Segment Routing, and they do not participate in BGP or PCEP. These core routers are ***Cisco IOS XRv 9000 7.2.2,*** which are named as:

- tinaa-vlab-gw-01
- tinaa-vlab-ls-01~04

Our edge routers have BGP neighborship with the route-reflector, and therefore ODL can have their link-state information. All three edge routers are configured as PCC nodes and peer with the ODL as our PCE. These routers are ***Nokia SROS-21.10.R2*** and named as:

- tinaa-vlab-se-01~03

The below diagram shows the details of our network and IP addresses in it.

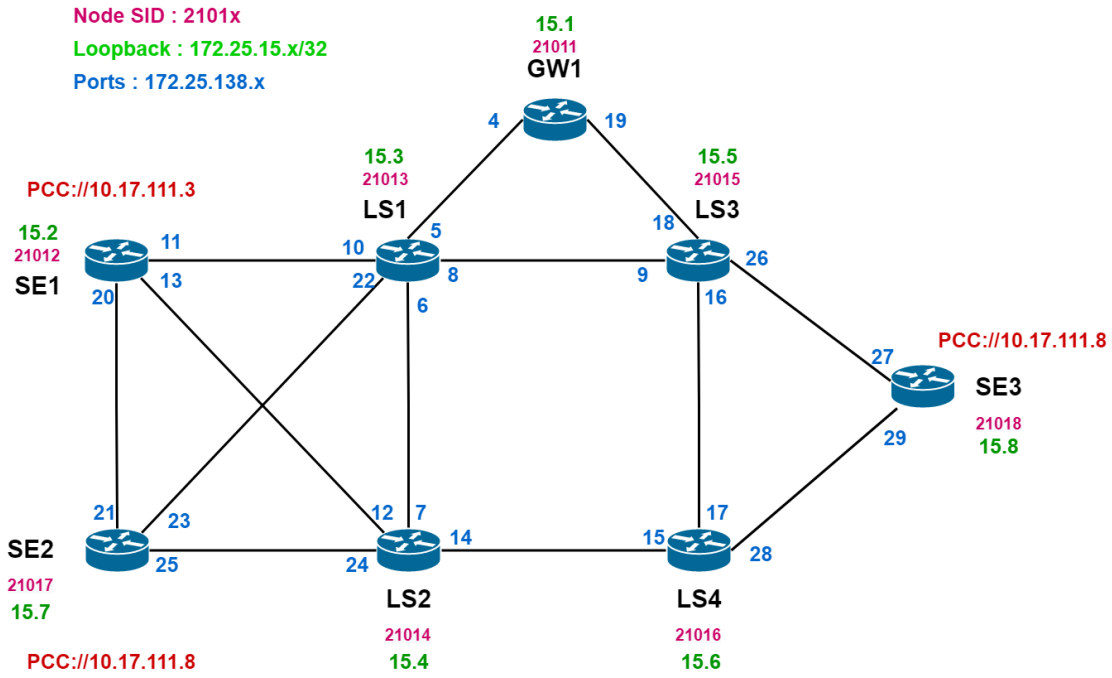


Figure 5-3 Network IP and SID details

Table 5-1 Routers' details

Router	Type	Loopback IP	Node SID
GW1	Core	172.25.15.1	21011
LS1	Core	172.25.15.3	21013
LS2	Core	172.25.15.4	21014
LS3	Core	172.25.15.5	21015
LS4	Core	172.25.15.6	21016
SE1	Edge	172.25.15.2	21012
SE2	Edge	172.25.15.7	21017
SE3	Edge	172.25.15.8	21018

In following subsections, we review the configuration of routers. For brevity, we only review the essential parts of the configurations.

5.2.1 Core Routers Configuration

Here we review the configuration for LS1. Other routers have the same configuration only with different IP addresses.

LS1 Configuration:

```
hostname tinaa-vlab-ls-01
!
vrf MGMT
 address-family ipv4 unicast
!
interface Loopback0
 ipv4 address 172.25.15.3 255.255.255.255
!
router static
 vrf MGMT
  address-family ipv4 unicast
   0.0.0.0/0 10.17.111.1
  !
 !
!
router isis LABCORE
 is-type level-2-only
 net 47.0416.1720.2501.5003.00
 segment-routing global-block 21011 199999
 log adjacency changes
 address-family ipv4 unicast
  metric-style wide
  advertise passive-only
  mpls traffic-eng level-2-only
  mpls traffic-eng router-id Loopback0
  mpls traffic-eng multicast-intact
  router-id Loopback0
  segment-routing mpls
 !
 interface Loopback0
  passive
  circuit-type level-2-only
  address-family ipv4 unicast
   prefix-sid index 2
  !
 !
 interface GigabitEthernet0/0/0/0.100
  circuit-type level-2-only
  point-to-point
```

```
 hello-password hmac-md5 encrypted 15060202052B
 address-family ipv4 unicast
  fast-reroute per-prefix
  fast-reroute per-prefix ti-lfa
  metric 1000
  mpls ldp sync
 !
!
interface GigabitEthernet0/0/0/1.100
 circuit-type level-2-only
 point-to-point
 hello-password hmac-md5 encrypted 111D100B1613
 address-family ipv4 unicast
  fast-reroute per-prefix
  fast-reroute per-prefix ti-lfa
  metric 1000
  mpls ldp sync
 !
!
interface GigabitEthernet0/0/0/2.100
 circuit-type level-2-only
 point-to-point
 hello-password hmac-md5 encrypted 02120D550A07
 address-family ipv4 unicast
  fast-reroute per-prefix
  fast-reroute per-prefix ti-lfa
  metric 1000
  mpls ldp sync
 !
!
interface GigabitEthernet0/0/0/3.100
 circuit-type level-2-only
 point-to-point
 hello-password hmac-md5 encrypted 03105205070E
 address-family ipv4 unicast
  fast-reroute per-prefix
  fast-reroute per-prefix ti-lfa
  metric 1000
  mpls ldp sync
 !
!
interface GigabitEthernet0/0/0/4.100
 circuit-type level-2-only
 point-to-point
 hello-password hmac-md5 encrypted 13111E1C0A0D
 address-family ipv4 unicast
  fast-reroute per-prefix
  fast-reroute per-prefix ti-lfa
  metric 1000
  mpls ldp sync
 !
!
```

```
!
mpls oam
!
mpls traffic-eng
 interface GigabitEthernet0/0/0/0.100
 !
 interface GigabitEthernet0/0/0/1.100
 !
 interface GigabitEthernet0/0/0/2.100
 !
 interface GigabitEthernet0/0/0/3.100
 !
 interface GigabitEthernet0/0/0/4.100
 !
!
```

Obviously, nothing special is going here. Interfaces are configured with proper IP addresses and we configured the ISIS routing. In the ISIS section, we shall determine the SRGB which is 21011 to 199999 for our case. This range is the same for whole network.

```
segment-routing global-block 21011 199999
```

Also, we need to enable the segment routing for MPLS by using below command:

```
segment-routing mpls
```

We use the loopback0 as the router ID and we need to assign it a Node SID. We assign it by index which means we determine the index of the Node SID in the range of SRGB. For LS1 we assigned the index of **2**. Therefore, the Node SID for this router would be **21013**. In the Table 5-1 you can find the same details for other routers.

The last main part is to enable MPLS Traffic Engineering for the interfaces.

5.2.2 Edge Routers Configuration

For edge routers we use ***Nokia SROS-21.10.R2*** as SE1, SE2, ad SE3.

SE1 Configuration:

```
#--------------------------------------------------
echo "Router (Network Side) Configuration"
#--------------------------------------------------
   router Base
      interface "system"
         address 172.25.15.2/32
         description "Loopback"
         icmp
            no mask-reply
            no redirects
         exit
         no shutdown
      exit
      interface "tinna-vlab-ls-01"
         address 172.25.138.11/31
         description "<< tinna-vlab-ls-01 | Gi0/0/0/3 | VLAN 100 >>"
         enable-ingress-stats
         ldp-sync-timer 15
         port 1/1/c1/1:100
         icmp
            no mask-reply
            no redirects
         exit
         no shutdown
      exit
      interface "tinna-vlab-ls-02"
         address 172.25.138.13/31
         description "<< tinna-vlab-ls-02 | Gi0/0/0/3 | VLAN 100 >>"
         enable-ingress-stats
         ldp-sync-timer 15
         port 1/1/c2/1:100
         icmp
            no mask-reply
            no redirects
         exit
         no shutdown
      exit
      interface "tinna-vlab-se-02"
         address 172.25.138.20/31
         description "<< tinna-vlab-se-02 | 1/1/c3/1 | VLAN 100 >>"
         enable-ingress-stats
         ldp-sync-timer 15
         port 1/1/c3/1:100
         icmp
            no mask-reply
            no redirects
         exit
         no shutdown
      exit
#--------------------------------------------------
```

```
echo "MPLS Label Range Configuration"
#--------------------------------------------------
    mpls-labels
      sr-labels start 21011 end 199999
    exit
#--------------------------------------------------
echo "ISIS Configuration"
#--------------------------------------------------
    isis 0
      level-capability level-2
      area-id 49.0416
      lsp-lifetime 65535
      lsp-refresh-interval 32767 half-lifetime enable
      overload-on-boot timeout 260
      traffic-engineering
      advertise-passive-only
      advertise-router-capability as
      loopfree-alternates
        ti-lfa
        exit
      exit
      timers
        spf-wait 1000 spf-initial-wait 10 spf-second-wait 50
      exit
      level 2
        wide-metrics-only
      exit
      segment-routing
        prefix-sid-range global
        tunnel-table-pref 8
        no shutdown
      exit
      interface "system"
        level-capability level-2
        ipv4-node-sid index 1
        passive
        no shutdown
      exit
      interface "tinna-vlab-ls-01"
        level-capability level-2
        interface-type point-to-point
        level 2
          hello-authentication-key "pI1XgPvzajgvVGC0LY4G0b7p08fZ" hash2
          hello-authentication-type message-digest
          hello-interval 5
          metric 1000
        exit
        no shutdown
      exit
      interface "tinna-vlab-ls-02"
        level-capability level-2
        interface-type point-to-point
```

```
            level 2
               hello-authentication-key "pI1XgPvzajgvVGC0LY4G0czp1wAf" hash2
               hello-authentication-type message-digest
               hello-interval 5
               metric 1000
            exit
            no shutdown
         exit
         interface "tinna-vlab-se-02"
            level-capability level-2
            interface-type point-to-point
            level 2
               hello-authentication-key "pI1XgPvzajgvVGC0LY4G0S3uREdi" hash2
               hello-authentication-type message-digest
               hello-interval 5
               metric 1000
            exit
            no shutdown
         exit
         no shutdown
      exit
#--------------------------------------------------
echo "Pcep Configuration"
#--------------------------------------------------
      pcep
         pcc
            local-address 10.17.111.3
            peer 10.16.6.11
               no shutdown
            exit
            no shutdown
         exit
      exit
#--------------------------------------------------
echo "MPLS Configuration"
#--------------------------------------------------
      mpls
         interface "system"
            no shutdown
         exit
         interface "tinna-vlab-ls-01"
            no shutdown
         exit
         interface "tinna-vlab-ls-02"
            no shutdown
         exit
         interface "tinna-vlab-se-02"
            no shutdown
         exit
      exit
#--------------------------------------------------
echo "RSVP Configuration"
```

```
#--------------------------------------------------
    rsvp
      interface "system"
        no shutdown
      exit
      interface "tinna-vlab-ls-01"
        refresh-reduction
          reliable-delivery
        exit
        no shutdown
      exit
      interface "tinna-vlab-ls-02"
        refresh-reduction
          reliable-delivery
        exit
        no shutdown
      exit
      interface "tinna-vlab-se-02"
        refresh-reduction
          reliable-delivery
        exit
        no shutdown
      exit
      no shutdown
    exit
#--------------------------------------------------
echo "MPLS LSP Configuration"
#--------------------------------------------------
    mpls
      path "DYNAMIC-CSPF"
        no shutdown
      exit
      path "pce-init"
        no shutdown
      exit
      path "my-ip-path"
        hop 1 172.25.15.4 strict
        hop 2 172.25.15.3 strict
        hop 3 172.25.15.7 strict
        no shutdown
      exit
      path "my-sid-path"
        hop 1 sid-label 21014
        hop 2 sid-label 21013
        hop 3 sid-label 21017
        no shutdown
      exit
      lsp-template "pce-init-template" pce-init-p2p-srte template-id default
        default-path "pce-init"
        max-sr-labels 7 additional-frr-labels 2
        pce-report enable
        no shutdown
```

```
        exit
        lsp "se01-se03-sr" sr-te
          to 172.25.15.8
          max-sr-labels 6 additional-frr-labels 2
          pce-report enable
          pce-control
          primary "DYNAMIC-CSPF"
          exit
          no shutdown
        exit
        pce-initiated-lsp
          sr-te
            no shutdown
          exit
        exit
        no shutdown
      exit
#--------------------------------------------------
echo "LDP Configuration"
#--------------------------------------------------
      ldp
        fast-reroute
        no shortcut-transit-ttl-propagate
        import-pmsi-routes
        exit
        session-parameters
          peer 172.25.15.3
          exit
          peer 172.25.15.4
          exit
          peer 172.25.15.7
          exit
        exit
        tcp-session-parameters
          peer-transport 172.25.15.3
            authentication-key "pI1XgPvzajgvVGC0LY4G0YGNLngu" hash2
          exit
          peer-transport 172.25.15.4
            authentication-key "pI1XgPvzajgvVGC0LY4G0WXNChwr" hash2
          exit
          peer-transport 172.25.15.7
            authentication-key "pI1XgPvzajgvVGC0LY4G0a56htJg" hash2
          exit
        exit
        interface-parameters
          interface "tinna-vlab-ls-01" dual-stack
            ipv4
              no shutdown
            exit
            no shutdown
          exit
          interface "tinna-vlab-ls-02" dual-stack
```

```
            ipv4
               no shutdown
            exit
            no shutdown
         exit
         interface "tinna-vlab-se-02" dual-stack
            ipv4
               no shutdown
            exit
            no shutdown
         exit
      exit
      targeted-session
      exit
      no shutdown
   exit
exit

#------------------------------------------------
echo "BGP Configuration"
#------------------------------------------------
   bgp
      multi-path
         maximum-paths 2
      exit
      ibgp-multipath
      enable-inter-as-vpn
      local-as 65038
      router-id 172.25.15.2
      enable-peer-tracking
      rapid-withdrawal
      rapid-update l2-vpn mvpn-ipv4
      best-path-selection
         origin-invalid-unusable
      exit
      next-hop-resolution
         shortcut-tunnel
            family ipv4
               resolution-filter
                  ldp
                  rsvp
               exit
               resolution filter
            exit
         exit
         labeled-routes
            transport-tunnel
               family label-ipv6
                  resolution-filter
                     ldp
                     rsvp
                     bgp
```

```
            exit
            resolution any
          exit
        exit
      exit
    exit
    rib-management
      ipv6
        route-table-import "SE-EXPORT-IPV6-VLAB-SE-01"
      exit
      label-ipv6
        route-table-import "SE-EXPORT-IPV6-VLAB-SE-01"
      exit
    exit
    group "ROUTE-REFLECTOR-VPN"
      description "<< Appliance Route Reflectors | VPN >>"
      min-route-advertisement 5
      next-hop-self
      peer-as 65038
      neighbor 172.25.212.13
        description "<< EDTNABTFRR43-ARR | AS65038 | ROUTE-REFLECTOR-VPN >>"
        family vpn-ipv4 vpn-ipv6 l2-vpn mvpn-ipv4 evpn
        authentication-key "pI1XgPvzajgvVGC0LY4G0X2yJhST" hash2
        keepalive 30
        hold-time 90
      exit
      neighbor 172.25.212.118
        description "<< EDTNABTFRR42-ARR | AS65038 | ROUTE-REFLECTOR-VPN >>"
        family vpn-ipv4 vpn-ipv6 l2-vpn mvpn-ipv4 evpn
        authentication-key "pI1XgPvzajgvVGC0LY4G0bNGjDbE" hash2
        keepalive 30
        hold-time 90
      exit
    exit
    no shutdown
  exit
exit
```

First, normal interface configuration and IP address assignment has been done. Then we determined the SRGB range for here too.

In the ISIS configuration, besides the usual configuration we need to enable traffic engineering, and segment routing. We tell the router to use the global range of prefix SIDs. For the loopback which is “system” interface here we assigned the index **1** for **ipv4-node-sid** which results to **21012**.

Unlike core routers, edge routers are PCC nodes and they should be configured for it. In the PCEP section, we configured the router as a PCC and gave it a unique IP address for communication with PCE peer which is the Opendaylight controller. The IP address of ODL in **10.16.6.11**.

The MPLS interface configuration is normal and nothing especial is going on here.

Then we reach to the "MPLS LSP CONFIGURATION" where we need to prepare the router for accepting the PCE Initiated LSPs.

First, we need to prepare a template for PCE initiated LSPs. The only supported type is **pce-init-p2p-srte**. We configure a default template while it is possible to have different templates by using different template IDs. We use a default path named **"pce-init"** just to satisfy the formalities. This path is just an empty path. For this template we enabled the reporting to PCE feature so we can monitor the status of the LSP.

In the **pce-initiated-lsp** we need to enable the SR-TE by using no shutdown command.

The following is the procedure for configuring and programming a PCE-initiated SR-TE LSP [26].

5.2.2.1 PCE-Initiated LSPs

1. The user must enable **pce-initiated-lsp sr-te** using the CLI.The user can also optionally configure a limit to the number of PCE-Initiated LSPs that the PCE can instantiate on a node using the **max-srte-pce-init-lsps** command in the CLI.
2. The user must configure at least one LSP template of type **pce-init-p2p-srte** to select the value of the LSP parameters that remain under the control of the PCC.

At a minimum, a **default** template should be configured (type pce-init-p2p-srte default). In addition, LSP templates with a defined template ID can be configured. The template ID can be included in the path profile of the PCInitiate message to indicate which non-default template to use for a particular LSP. If the PCInitiate message does not include the PCE path profile, MPLS uses the default PCE-initiated LSP template.

5.2.2.2 Application Generation of PCInitiate

Below is the procedure that Nokia is defined when an application, in our case ODL, wants to initiate an LSP:

1. When the PCEP session is established from the PCC to PCE, the PCC and PCE exchange the Open object and both set the new "I flag, LSP-INSTANTIATION CAPABILITY" flag, in the STATEFUL-PCE-CAPABILITY TLV flag field.
2. The operator, using the north-bound REST interface, makes a request to the ODL to initiate an LSP. The following parameters are specified:
    a. source address
    b. destination address
    c. LSP type (SR-TE)
    d. bandwidth value
    e. include/exclude admin-group constraints
    f. optional PCE path profile ID for the path computation at the PCE
    g. optional PCE-initiated LSP template ID for use by the PCC to complete the instantiation of the LSP
3. The ODL crafts the PCInitiate message and sends it to the PCC using PCEP. The message contains the LSP object with PLSP-ID=0, the SRP object, the ENDPOINTS object, the computed SR-ERO (SR-TE) object, and the list of LSP attributes (bandwidth object, one or more metric objects, and the LSPA object). The LSP path name is inserted into the Symbolic Path Name TLV in the LSP object.
4. The PCE-initiated LSP template ID to be used at the PCC, if any, is included in the PATH-PROFILE-ID TLV of the Path Profile object. The Profile ID matches the PCE-initiated LSP template ID at the PCC and is not the same as The Path Profile ID is used on the PCE to compute the path of this PCE-initiated LSP.
5. The Path Profile ID is used on the PCE to compute the path of this PCE-initiated LSP.

5.2.2.3 SR OS Router Procedures on Receiving a PCInitiate Message

1. If a PCInitiate message includes a name that is a duplicate of an existing LSP on the router, the system generates an error.
2. The router assigns a PLSP-ID and looks up the specified PCE-initiated LSP template ID, if any, or the default PCE-initiated LSP template, to retrieve the local parameters, and instantiates the SR-TE LSP.

3. The instantiated LSP is added to the database and is used by all applications that look up a tunnel in the database.
4. The router crafts a PCRpt message with the Tunnel-ID, LSP-ID, and the RRO and passes it along with the PLSP-ID set to the assigned value and the delegation bit set in the LSP object to the PCE.

5.3 Opendaylight Installation

We use the OpenDayLight Silicon-SR2 version. For complete guide you can refer to [27].

The default distribution can be found on the OpenDaylight software download page:

https://docs.opendaylight.org/en/latest/downloads.html

Before using ODL, we need to install JAVA OpenDaylight requires a Java runtime environment (JRE) to run. OpenDaylight can leverage either a stand-alone JRE on the JRE bundled in a Java Software Development Kit.

The following command installs the JAVA 11 JRE.

```
$ sudo apt-get -y install openjdk-8-jre
```

Then we need to retrieve the full path to JAVA executable:

```
$ ls -l /etc/alternatives/java
lrwxrwxrwx 1 root root 43 Nov 22 20:48 /etc/alternatives/java -> /usr/lib/jvm/java-11
-openjdk-amd64/bin/java
```

Then we set JAVA_HOME by running the next command:

```
$ export JAVA_HOME= /usr/lib/jvm/java-11-openjdk-amd64
```

The Karaf distribution has no features enabled by default. However, all of the features are available to be installed.

To run the Karaf distribution:

1. Unzip the zip file.
2. Navigate to the directory.
3. run ./bin/karaf.

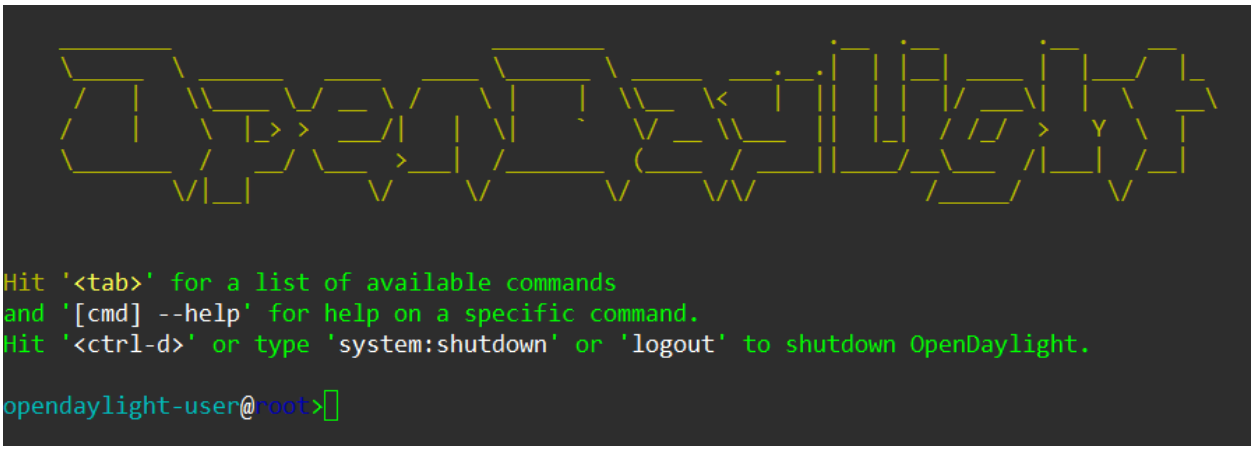


Figure 5-4 First run of ODL

** Although, above picture is just to show the ODL command line environment and we use a docker container for ODL. This way every time we run the container, we can have a clear system with basic configurations already done.*

- Press tab for a list of available commands
- Typing [cmd] --help will show help for a specific command.
- Press ctrl-d or type system:shutdown or logout to shutdown OpenDaylight.

For installing the Karaf features:

```
feature:install <feature1>
```

You can install multiple features using the following command:

```
feature:install <feature1> <feature2> ... <featureN-name>
```

For listing available features:

feature:list Lists all existing features available from the defined repositories.

feature:list -i List only installed features

feature:list -i | grep List the installed features and grep with specific name.

Now its time to install the required features for our work.

- feature:install **odl-mdsal-apidocs**
- feature:install **odl-restconf**
- feature:install **odl-bgpcep-bgp**
- feature:install **odl-bgpcep-pcep**

with above feature installed we have all we need to start working with ODL.

5.4 ODL on a Docker Container

All said above and required configurations already done and packed as a docker container. Our ODL container is running a lab server and in below figure you can see the port mapping of for our container.

```
[mostafa@tinaa-client-2 ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS      PORTS

                            NAMES
6b0ee866492b   odl       "/opt/odl/bin/karaf …"   2 months ago   Up 5 days   0.0.0.0:179->179/tcp, :::179->
179/tcp, 0.0.0.0:4189->4189/tcp, :::4189->4189/tcp, 0.0.0.0:8185->8185/tcp, :::8185->8185/tcp, 0.0.0.0:80->
8181/tcp, :::80->8181/tcp   epic_bell
[mostafa@tinaa-client-2 ~]$
```

Figure 5-5 ODL running docker container

5.5 NorthBound Interface - RestConf/Rest API

Standard mechanisms to allow Web applications to access the configuration data, state data, data-model-specific Remote Procedure Call (RPC) operations, and event notifications within a networking device, in a modular and extensible manner.

RESTCONF uses HTTP methods to provide CRUD operations on a conceptual datastore containing YANG-defined data, which is compatible with a server that implements NETCONF datastores.

The RESTCONF protocol uses HTTP methods to identify the CRUD operations requested for a particular resource.

Following are the widely used methods:

GET:

The GET method is sent by the client to retrieve data and metadata for a resource. It is supported for all resource types, except operation resources.

If the user is not authorized to read the target resource, an error response containing a "401 Unauthorized" status-line SHOULD be returned.

PUT:

The PUT method is sent by the client to create or replace the target data resource. A request message-body MUST be present, representing the new data resource.

The "insert" and "point" query parameters MUST be supported by the PUT method for data resources.

If the PUT request creates a new resource, a "**201** Created" status-line is returned. If an existing resource is modified, a "**204** No Content" status-line is returned.

DELETE:

The DELETE method is used to delete the target resource. If the DELETE request succeeds, a "**204** No Content" status-line is returned. If the user is not authorized to delete the target resource, then an error response containing a "**403** Forbidden" status-line SHOULD be returned.

POST:

The POST method is sent by the client to create a data resource or invoke an operation resource.

5.6 OpenDaylight PCEP plugin

The OpenDaylight PCEP plugin provides all basic service units necessary to build-up a PCE-based controller. In addition, it offers LSP management functionality for Active Stateful PCE - the cornerstone for majority of PCE-enabled SDN solutions. It consists of the following components:

- Protocol library
- PCEP session handling
- Stateful PCE LSP-DB
- Active Stateful PCE LSP Operations

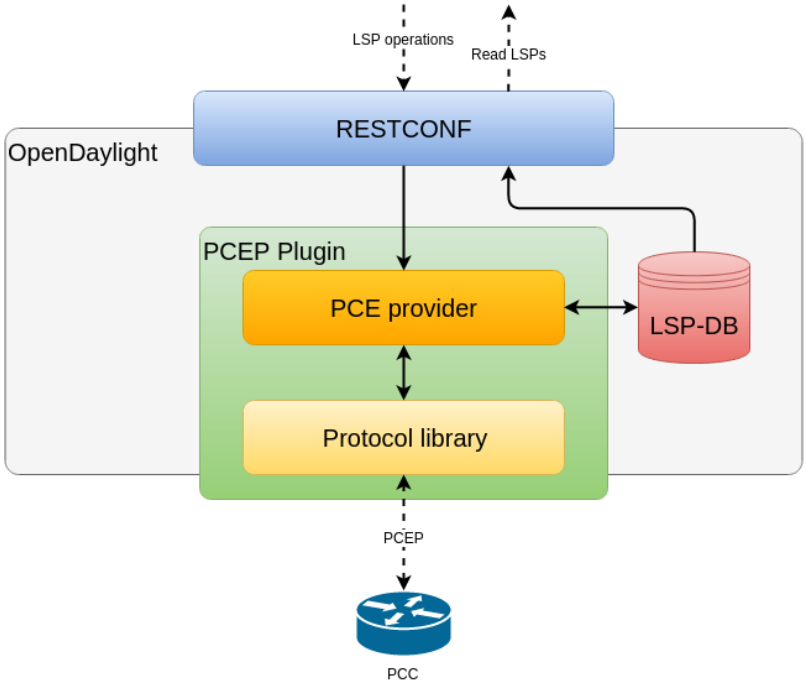


Figure 5-6 OpenDaylight PCEP plugin overview

As mentioned before to use the PCEP with ODL we need to install the **odl-bgpcep-pcep** feature:

```
feature:install odl-restconf odl-bgpcep-pcep
```

The PCEP plugin contains a default configuration, which is applied after the feature starts up. One instance of PCEP plugin is created (named **pcep-topology**), and its presence can be verified via REST:

URL: `restconf/operational/network-topology:network-topology/topology/pcep-topology`

Method: `GET`

Actually, above API returns the pcep-topology of the network.

```
{
    "topology": [
        {
            "topology-id": "pcep-topology",
            "topology-types": {
                "network-topology-pcep:topology-pcep": {}
            },
            "node": [
                {
                    "node-id": "pcc://172.25.130.73",
                    "network-topology-pcep:path-computation-client": {
                        "state-sync": "synchronized",
                        "ip-address": "172.25.130.73",
                        "stateful-tlv": {
                            "odl-pcep-ietf-stateful:stateful": {
                                "lsp-update-capability": true,
                                "odl-pcep-ietf-initiated:initiation": true
                            }
                        },
```


Figure 5-7 Sample result of getting pcep-topology from the ODL

Please note that this capability is enabled by default. No additional configuration is required.

The LSP State Database (LSP-DB) contains an information about all LSPs and their attributes. The LSP state is synchronized between the PCC and PCE. First, initial LSP state synchronization is performed once the session between PCC and PCE is established in order to learn PCC's LPSs. This step is a prerequisite to following LSPs manipulation operations.

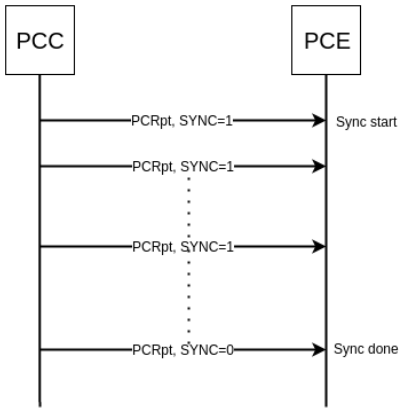


Figure 5-8 LSP State Synchronization.

5.6.1 LSP-DB API

Below figure shows a part of LSP-DB, since it's a large structure, for complete structure please refer to [28].

```
path-computation-client
   +--ro reported-lsp* [name]
      +--ro name        string
      +--ro path* [lsp-id]
      |  +--ro lsp-id                     rsvp:lsp-id
      |  +--ro ero
      |  |  +--ro processing-rule?   boolean
      |  |  +--ro ignore?            boolean
      |  |  +--ro subobject*
      |  |     +--ro loose         boolean
      |  |     +--ro (subobject-type)?
      |  |        +--:(as-number-case)
      |  |        |  +--ro as-number
      |  |        |     +--ro as-number    inet:as-number
      |  |        +--:(ip-prefix-case)
      |  |        |  +--ro ip-prefix
      |  |        |     +--ro ip-prefix    inet:ip-prefix
      |  |        +--:(label-case)
      |  |        |  +--ro label
      |  |        |     +--ro uni-directional            boolean
      |  |        |     +--ro (label-type)?
      |  |        |        +--:(type1-label-case)
      |  |        |        |  +--ro type1-label
      |  |        |        |     +--ro type1-label    uint32
      |  |        |        +--:(generalized-label-case)
```

Figure 5-9 PCC in LSP-DB [28]

The LSP-DB is accessible via RESTCONF. The PCC's LSPs are stored in the pcep-topology while the session is active. In a next example, there is one PCEP session with PCC identified by its IP address (10.17.111.3) and one reported LSP ("se01-se03-sr::DYNAMIC-CSPF").

URL: `/restconf/operational/network-topology:network-topology/topology/pcep-topology/node/pcc:%2F%2F10.17.111.3`

Method: `GET`


```
{
    "node": [
        {
            "node-id": "pcc://10.17.111.3",
            "network-topology-pcep:path-computation-client": {
                "state-sync": "synchronized",
                "ip-address": "10.17.111.3",
                "stateful-tlv": {
                    "odl-pcep-ietf-stateful:stateful": {
                        "lsp-update-capability": true,
                        "odl-pcep-ietf-initiated:initiation": true
                    }
                },
                "reported-lsp": [
                    {
                        "name": "se01-se03-sr::DYNAMIC-CSPF",
                        "path": [
                            {
                                "lsp-id": 7168,
```


Figure 5-10 Part of pcep-topology for node SE1

5.6.2 PCEP Extensions for Segment Routing

The PCEP Extensions for Segment Routing (SR) allow a stateful PCE to compute and initiate TE paths in SR networks. The SR path is defined as an order list of segments. Segment Routing architecture can be directly applied to the MPLS forwarding plane without changes. Segment Identifier (SID) is encoded as a MPLS label.

This capability is enabled by default.

The PCEP SR extension defines new ERO subobject - SR-ERO subobject capable of carrying a SID.

```
sr-ero-type
   +---- c-flag?               boolean
   +---- m-flag?               boolean
   +---- sid-type?             sid-type
   +---- sid?                  uint32
   +---- (nai)?
      +--:(ip-node-id)
      |  +---- ip-address            inet:ip-address
      +--:(ip-adjacency)
      |  +---- local-ip-address      inet:ip-address
      |  +---- remote-ip-address     inet:ip-address
      +--:(unnumbered-adjacency)
         +---- local-node-id         uint32
         +---- local-interface-id    uint32
         +---- remote-node-id        uint32
         +---- remote-interface-id   uint32
```

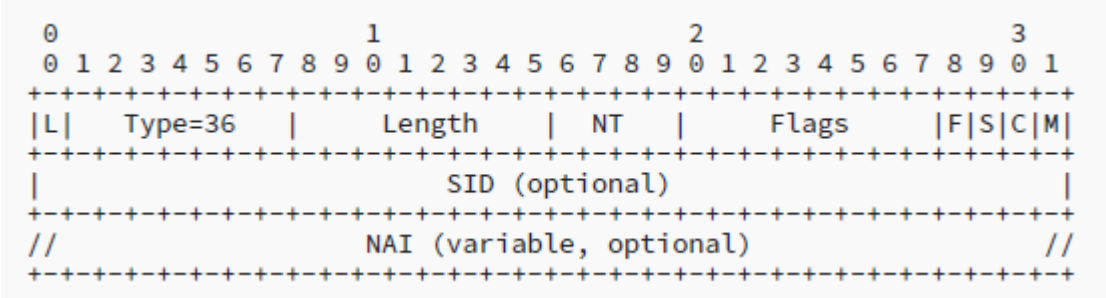


Figure 5-11 SR-ERO Subobject Format - RFC 8664

Based of the RFC 8664, the fields in the SR-ERO subobject are as follows:

L-Flag:

Indicates whether the subobject represents a loose hop in the LSP [RFC3209]. If this flag is set to zero, a PCC MUST NOT overwrite the SID value present in the SR-ERO subobject. Otherwise, a PCC MAY expand or replace one or more SID values in the received SR-ERO based on its local policy.

Type:

Set to 36.

Length:

Contains the total length of the subobject in octets. The Length MUST be at least 8 and MUST be a multiple of 4. An SR-ERO subobject MUST contain at least one SID or NAI. The flags described below indicate whether the SID or NAI fields are absent.

NAI Type (NT):

Indicates the type and format of the NAI contained in the object body, if any is present. If the F bit is set to zero (see below), then the NT field has no meaning and MUST be ignored by the receiver. This document describes the following NT values:

NT=0 The NAI is absent.
NT=1 The NAI is an IPv4 node ID.
NT=2 The NAI is an IPv6 node ID.
NT=3 The NAI is an IPv4 adjacency.
NT=4 The NAI is an IPv6 adjacency with global IPv6 addresses.
NT=5 The NAI is an unnumbered adjacency with IPv4 node IDs.
NT=6 The NAI is an IPv6 adjacency with link-local IPv6 addresses.

Flags:

Used to carry additional information pertaining to the SID. This document defines the following flag bits. The other bits MUST be set to zero by the sender and MUST be ignored by the receiver.

M:

If this bit is set to 1, the SID value represents an MPLS label stack entry as specified in [RFC3032]. Otherwise, the SID value is an administratively configured value that represents an index into an MPLS label space (either SRGB or SRLB) per [RFC8402].

C:

If the M bit and the C bit are both set to 1, then the TC, S, and TTL fields in the MPLS label stack entry are specified by the PCE. However, a PCC MAY choose to override these values according to its local policy and MPLS forwarding rules. If the M bit is set to 1 but the C bit is set to zero, then the TC, S, and TTL fields MUST be ignored by the PCC. The PCC MUST set these fields according to its local policy and MPLS forwarding rules. If the M bit is set to zero, then the C bit MUST be set to zero.

S:

When this bit is set to 1, the SID value in the subobject body is absent. In this case, the PCC is responsible for choosing the SID value, e.g., by looking it up in the SR-DB using the NAI that, in this case, MUST be present in the subobject. If the S bit is set to 1, then the M and C bits MUST be set to zero.

F:

When this bit is set to 1, the NAI value in the subobject body is absent. The F bit MUST be set to 1 if NT=0; otherwise, it MUST be set to zero. The S and F bits MUST NOT both be set to 1.

SID:

The Segment Identifier. Depending on the M bit, it contains either:

* A 4-octet index defining the offset into an MPLS label space per [RFC8402] or

* A 4-octet MPLS label stack entry, where the 20 most significant bits encode the label value per [RFC3032].

NAI:

The NAI associated with the SID. The NAI's format depends on the value in the NT field and is described in the following section.

At least one SID and NAI MUST be included in the SR-ERO subobject, and both MAY be included.

The RFC 8664 document defines the following NAIs:

IPv4 Node ID:

Specified as an IPv4 address. In this case, the NT value is 1, and the NAI field length is 4 octets.

IPv4 Adjacency:

Specified as a pair of IPv4 addresses. In this case, the NT value is 3, and the NAI field length is 8 octets. The format of the NAI is shown in the following figure:

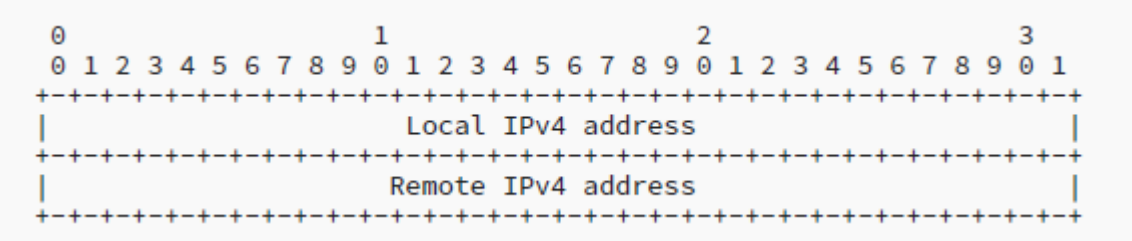


Figure 5-12 NAI for IPv4 Adjacency

Considering all above, we have three main actions on PCE Initiated LSPs: **Add LSP**, **Update LSP**, and **Remove LSP**

5.6.3 LSP Operations for PCEP SR

5.6.3.1 Add LSP

Following RPC example illustrates a request for the SR-TE LSP creation:

URL: `/restconf/operations/network-topology-pcep:add-lsp`

Method: `POST`

Below example shows the body of our request to create a LSP from SE1 PCC named "PCE-Init-LSP-01" to the SE2 router. As you can see the "path-setup-type" is equal to 1 to select the segment routing.

Please notice that "m-flag" value is False so we use SID not MPLS labels.

```
{
    "input": {
        "node": "pcc://{{SE1-pcep-id}}",
        "name": "PCE-Init-LSP-01",
        "arguments": {
            "lsp": {
                "delegate": true,
                "administrative": true
            },
            "endpoints-obj": {
                "ipv4": {
                    "source-ipv4-address": "{{SE1-ip}}",
                    "destination-ipv4-address": "{{SE2-ip}}"
                }
            },
            "path-setup-type": {
                "pst": 1
            },
            "ero": {
                "subobject": {
                    "loose": false,
                    "nai-type": "ipv4-node-id",
                    "m-flag": false,
                    "sid": {{LS3-sid}},
                    "ip-address": "{{LS3-ip}}"
                }
            }
        },
        "network-topology-ref": "/network-topology:network-topology/network-
topology:topology[network-topology:topology-id=\"pcep-topology\"]"
    }
}
```

5.6.3.2 Update LSP

Following RPC example illustrates a request for the LSP update:

URL: `/restconf/operations/network-topology-pcep:update-lsp`

Method: `POST`

Below example shows the json data as the body of our request to update the previous LSP.

Since the endpoints of the LSP are the same, normally we don't have them in them in the body for update request.

As you can see, everything is them same except for **ero**, which is changed.

```
{
    "input": {
        "node": "pcc://{{SE1-pcep-id}}",
        "name": "PCE-Init-LSP-01",
        "arguments": {
            "lsp": {
                "delegate": true,
                "administrative": true
            },
            "path-setup-type": {
                "pst": 1
            },
            "ero": {
                "subobject": [
                    {
                        "loose": false,
                        "nai-type": "ipv4-node-id",
                        "m-flag" : false,
                        "sid": {{LS1-sid}},
                        "ip-address": "{{LS1-ip}}"
                    },
                    {
                        "loose": false,
                        "nai-type": "ipv4-node-id",
                        "m-flag" : false,
                        "sid": {{LS2-sid}},
                        "ip-address": "{{LS2-ip}}"
                    }
                ]
            }
        },
        "network-topology-ref": "/network-topology:network-topology/network-
topology:topology[network-topology:topology-id=\"pcep-topology\"]"
    }
}
```

5.6.3.3 Remove LSP

To remove an LSP below RPC is used.

URL: `/restconf/operations/network-topology-pcep:remove-lsp`

Method: `POST`

Below json data shows the body of our request to delete the LSP named " API-SE1-SE2-LSP-01 " from the SE1.

```
{
    "input": {
        "node": "pcc://{{SE1-node-id}}",
        "name": "API-SE1-SE2-LSP-01",
        "network-topology-ref": "/network-topology:network-topology/network-
topology:topology[network-topology:topology-id=\"pcep-topology\"]"
    }
}
```

Unfortunately, this option is not working in our experiment. Although everything has been checked till now, we couldn't solve this issue!

As per the OpenDayLight documentation [29] above request is all we need to use for deletion of an LSP in the PCC. Also, in [30] the procedures for deleting a PCE initiated LSP has been defined. Nokia uses its own controller named NSP. To delete the LSP, the NSP (in our case Opendaylight) crafts a PCInitiate message for the corresponding PLSP-ID and sets the R-bit in the SRP object flags to indicate to the PCC that it must delete the LSP. The NSP sends the message to the PCC using PCEP.

Opendaylight should do the same. To check we use Wireshark to examine the packets:

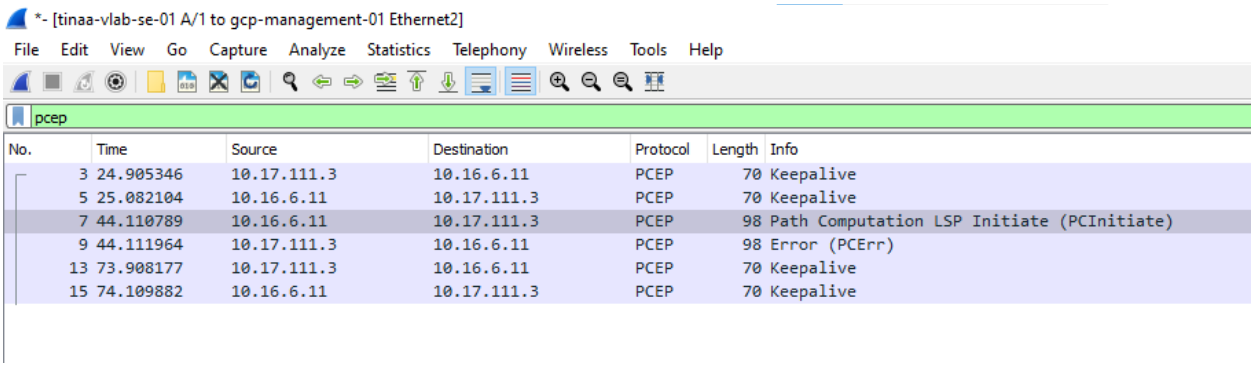


Figure 5-13Captured PCEP messages between ODL and SE1 (PCC)

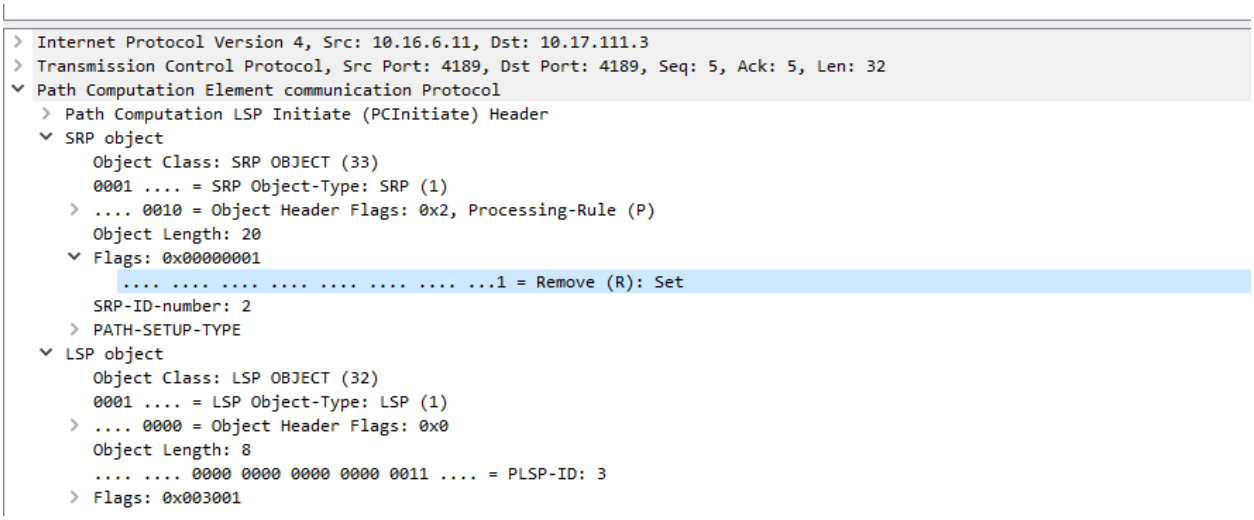


Figure 5-14 PCEP PCInitiate message from ODL to PCC

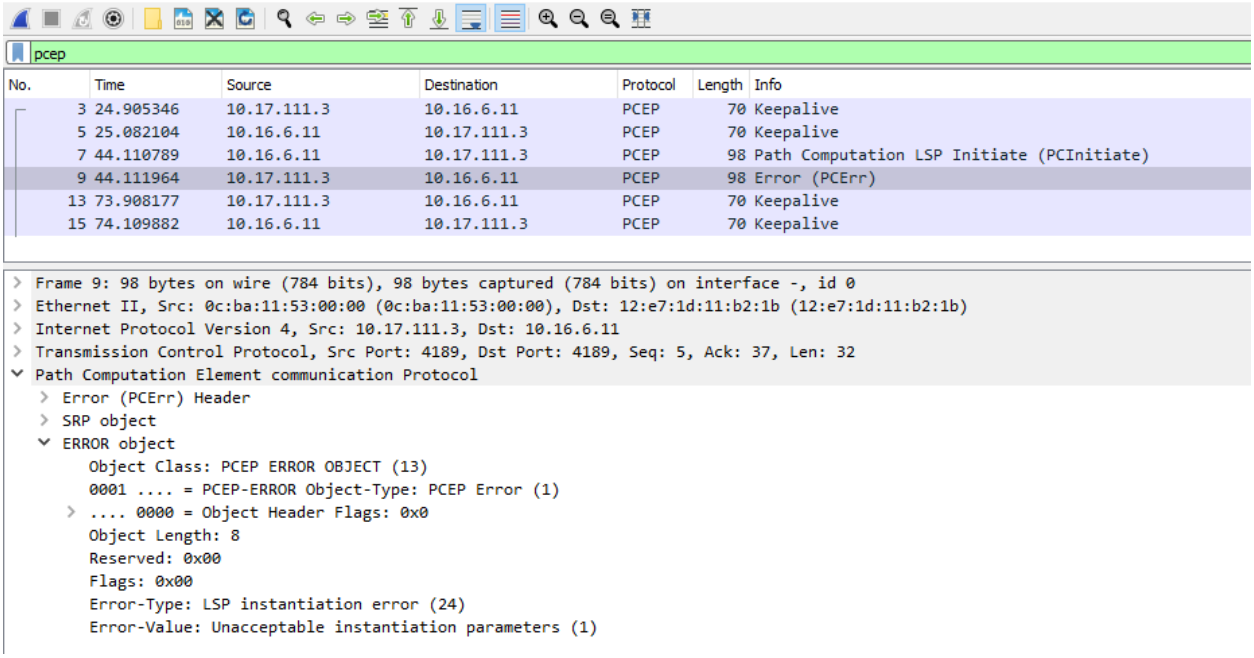


Figure 5-15 PCEPPCErr message from PCC to ODL

To make sure that ODL has sent the correct PLSP-ID we check the LSP-db by using below command:

show router pcep pcc lsp-db

```
PCEP-specific LSP ID: 3
LSP ID               : 11776            LSP Type            : pce-init-seg-rt
Tunnel ID            : 16386            Extended Tunnel Id  : 172.25.15.2
LSP Name             : API-SE1-SE2-LSP-01
Source Address       : 172.25.15.2      Destination Address : 172.25.15.7
LSP Delegated        : True             Delegate PCE Address: 10.16.6.11
Oper Status          : active
```

Figure 5-16 LSP Details in LSP-db on the SE1 Router (PCC)

Above image shows the part of the command result and obviously the PLSP-ID is 3.

The error ODL returns is:


```
{
    "output": {
        "error": [
            {
                "error-object": {
                    "processing-rule": false,
                    "ignore": false,
                    "value": 1,
                    "type": 24
                }
```

```
                }
            ],
            "failure": "failed"
        }
}
```

Checking the [31] we can find the error:

24	LSP instantiation error	0: Unassigned	[RFC8281]
		1: Unacceptable instantiation parameters	[RFC8281]
		2: Internal error	[RFC8281]
		3: Signaling error	[RFC8281]

Figure 5-17 Error type for LSP deletion request

5.7 Application

Before writing our simple program, we used Postman to get data and post our requests to the ODL. With using Postman, we can easily see the results of the requests and manage writing our program. After ensuring that we can move to writing our program.

Actually, this program consists of different functions which can be used for different creation, modification and monitoring proposes.

Using Flask, we added different routes to our program so it can be accessed through simple APIs and the ODL layer is hidden from the user.

Here, we explain all the functions from the low-level functions to perform the basic tasks to the higher-level functions.

5.7.1 odl_api_control Module

This module contains definition of all the constant variables, functions and customizable strings as URLs to build required requests.

By using this module, we can utilize our functions to perform different tasks.

As mentioned before, we use a docker container for our ODL and we showed how the ports are mapped to each other. To make it easier to pass the required ODL information, we defined them in a simple dictionary named odl_settings. It consists of required information to set the parameters to make a request to ODL.

```
odl_ip = '10.16.6.11'
odl_port = '80'
odl_user = 'admin'
odl_password = 'admin'
odl_pcep_topology_id = 'pcep-topology'
odl_bgpls_topology_id = 'lab-topology'

odl_settings = {
    'odl_ip': odl_ip,
    'odl_port': odl_port,
    'odl_user': odl_user,
    'odl_password': odl_password,
    'odl_pcep_topology_id': odl_pcep_topology_id,
    'odl_bgpls_topology_id': odl_bgpls_topology_id
}
```

Also, to create, update, and delete the LSPs we need to use below URLs when we make the requests:

```
create_lsp_url = 'http://%s:%s/restconf/operations/network-topology-pcep:add-lsp' %(odl_ip, odl_port)
update_lsp_url = 'http://%s:%s/restconf/operations/network-topology-pcep:update-lsp' %(odl_ip, odl_port)
delete_lsp_url = 'http://%s:%s/restconf/operations/network-topology-pcep:remove-lsp' %(odl_ip, odl_port)
```

For ease of working, we added our routers loopback IP address, Node-SID and PCC ID to this module.

5.7.2 get_url(url, headers, odl_settings)

The name of function is self-explanatory. We just pass the **url**, **headers**, and required settings to it and it gets the response. If the response is Ok then it returns it. Otherwise, it generates an error and print out the error.

This function checks the timeout exceptions as well as connection errors.

5.7.3 post_url(url, data, odl_settings)

```
def post_url(url, data, odl_settings):
    import requests
    response = requests.post(url=url, data=data, auth=(odl_settings['odl_user'], odl_settings['odl_password']), headers={'Content-Type': 'application/json'})
    return response
```

5.7.4 get_pcep_nodes(odl_settings)

There are two topologies available in our case:

- Network topology build upon BGP-LS information which consists of a list of **Links** and a list of **Nodes**
- PCEP topology that consists of **PCC Nodes** and their **Reported LSPs**.

This function gets the pcep topology and extract the list of nodes from it.

```
107  def get_pcep_nodes(odl_settings):
108
109      url = 'http://{}:{}/restconf/operational/network-topology:network-topology/topology/{}'
110      url = url.format(odl_settings['odl_ip'], odl_settings['odl_port'], odl_settings['odl_pcep_topology_id'])
111      headers={'content-type': 'application/json'}
112      result = get_url(url, headers, odl_settings)
113      if not result:
114          sys. exit()
115      pcep_topology = result.json()['topology'][0]
116      """
117          pcep_topology is a dict including:
118          topology-id : odl_pcep_topology_id
119          network-pcep-topology-config:session-config :
120          topology-types
121          node : list of nodes. Each node is a dict and each node at least has "node-id" key
122      """
123      pcep_nodes = pcep_topology['node']
124
125      return pcep_nodes
```

Elements of the node list are dictionaries and each one at least has a “node-id” key. For example:

```
"node-id": "pcc://10.17.111.8"
```

Or

```
"node-id": "10.17.111.3"
```

5.7.5 get_pcc_nodes(odl_settings)

To extract only PCC nodes we use this function. The **get_pcep_nodes** returns all nodes while we may need only PCC nodes. This function gets the nodes from previous one and extract those that have “pcc” in their node-id.

```
127  def get_pcc_nodes(odl_settings):
128      pcep_nodes = get_pcep_nodes(odl_settings)
129      pcc_nodes = []
130      for node in pcep_nodes:
131          # print(node['node-id'])
132          if node['node-id'][0:3] == "pcc":
133              pcc_nodes.append(node)
134      return pcc_nodes
```

5.7.6 get_pcc_node(odl_settings, node_id)

by passing the specific node-id, this function gets that PCC information.

If it is not available the result will be null.

5.7.7 get_reported_lsp(odl_settings, node_id)

This function returns the reported LSPs of a specific PCC.

```
146  def get_reported_lsp(odl_settings, node_id):
147      node =  get_pcc_node(odl_settings, node_id)
148      if "reported-lsp" in node["network-topology-pcep:path-computation-client"].keys():
149          reported_lsp = node["network-topology-pcep:path-computation-client"]["reported-lsp"]
150
151          '''
152              reported_lsp is a list of LSPs for each node. Each LSP is a dict containing following keys:
153                  "name": string
154                  "path" : a list where each item is a dict
155
156          '''
157          return reported_lsp
158      return
```

5.7.8 get_lsp_names(reported_lsp)

This function accepts a list of reported_lsp and extract the LSP names from it and return them as a list.

5.7.9 get_lsp(odl_settings, node_id, lsp_name)

Having the name of an LSP from a PCC we can get the information of that LSP by using this function.

```
166  def get_lsp(odl_settings, node_id, lsp_name):
167      reported_lsp = get_reported_lsp(odl_settings, node_id)
168      for ilsp in reported_lsp:
169          if ilsp['name'] == lsp_name:
170              return ilsp
171      return
```

5.7.10 get_lsp_path(odl_settings, node_id, lsp_name)

This function extracts the path from a reported LSP.

Below code is a piece of json data from an LSP

```
692                 "reported-lsp": [
693                     {
694                         "name": "se01-se03-sr::DYNAMIC-CSPF",
695                         "path": [
696                             {
697                                 "lsp-id": 7168,
698                                 "ero": {
699                                     "subobject": [
700                                         {
701                                             "loose": true,
702                                             "odl-pcep-segment-routing:m-flag": true,
703                                             "odl-pcep-segment-routing:nai-type": "ipv4-node-id",
704                                             "odl-pcep-segment-routing:c-flag": false,
705                                             "odl-pcep-segment-routing:sid": 21018,
706                                             "odl-pcep-segment-routing:ip-address": "172.25.15.8"
707                                         }
```

One of the keys in a reported is the “path” which is a list, usually with one element.

This element itself is a dictionary and one of its keys is “ero” which is a dictionary too and includes a list of subobjects. Actually, each element of subobject is a hop of the path.

The keys for SID and IP address of each hop are:

- "odl-pcep-segment-routing:sid"
- "odl-pcep-segment-routing:ip-address"

And if it is just an IP prefix:

- "ip-prefix"

By checking above keys it builds a path as a list with each item is a dictionary with two keys: 'ip-address' and 'sid'.

```
173  def get_lsp_path(odl_settings, node_id, lsp_name):
174
175      sid_key = "odl-pcep-segment-routing:sid"
176      ip_key = "odl-pcep-segment-routing:ip-address"
177      ip_prefix_key = "ip-prefix"
178
179      lsp = get_lsp(odl_settings, node_id, lsp_name)
180      hops = lsp["path"][0]["ero"]["subobject"]
181      lsp_path = []
182      for hop in hops:
183          sid = None
184          ip = None
185          if sid_key in hop.keys():
186              sid = hop[sid_key]
187          if ip_key in hop.keys():
188              ip = hop[ip_key]
189          if ip_prefix_key in hop.keys():
190              ip_prefix = hop[ip_prefix_key][ip_prefix_key]
191              ip = ip_prefix.split('/')[0]
192          lsp_path.append({ 'ip-address': ip, 'sid': sid})
193
194      return lsp_path
```

5.7.11 get_lsp_status(odl_settings, node_id, lsp_name, key)

To get the status of an LSP, this function returns the value for the passed key.

```
196  def get_lsp_status(odl_settings, node_id, lsp_name, key):
197      """
198          key can be any of following items:
199                                  "processing-rule"     : true / false
200                                  "remove"              : true / false
201                                  "sync"                : true / false
202                                  "administrative"      : true / false
203                                  "ignore"              : true / false
204                                  "plsp-id"             : int
205                                  "operational"         : "active" / "down"
206                                  "delegate"            : true / false
207      """
208      lsp = get_lsp(odl_settings, node_id, lsp_name)
209      if lsp:
210          return lsp["path"][0]["odl-pcep-ietf-stateful:lsp"][key]
211      return
```

Each LSP has a “path” key and usually it is a one item list which is a dict.

This dictionary has a “odl-pcep-ietf-stateful:lsp” key which contains the status information we need.

5.7.12 create_json_sr_lsp(src, dst, name_of_lsp , pcc, sr_path)

This function gets the required information of an LSP and generates a json structure ready for creating a LSP in the routers.

```
247     src : Source IP Address
248     dst : Destination IP Address
249     pcc : like "pcc://1.1.1.1"
250     path : a list of dicts each dict {'sid': 10, 'ip-address': '1.1.1.1', 'nai-type': 'ipv4-node-id' or 'ipv4-adjacency'}
```

The main part is putting the hops from the sr_path list into the json structure.

```
235     explicit = []
236     for hop in sr_path:
237         entry = {  "loose": "false",
238                    "nai-type": "ipv4-node-id",
239                    "m-flag": "false",
240                    "sid": hop['sid'],
241                    "ip-address": hop['ip-address']
242                 }
243         explicit.append(entry)
244         lsp_dict["input"]["arguments"]["ero"] = {"subobject":explicit}
245
246     return lsp_dict
```

We define an empty list named explicit and put each hop in it then we put this list into our lsp_dict and return it as our data to pass to ODL.

5.7.13 create_sr_lsp(odl_settings, src, dst, name_of_lsp , pcc, sr_path)

This function actually creates the LSP into our network.

```
285     sr_lsp_dict = create_json_sr_lsp(src, dst, name_of_lsp , pcc, sr_path)
286     response = post_url(create_lsp_url, json.dumps(sr_lsp_dict), odl_settings)
```

We use a/m function to create proper SR-LSP structure then we pass it to the post_url function with **create_lsp_url**.

5.7.14 create_json_update_sr_lsp(name_of_lsp , pcc, sr_new_path)

This function acts like create_json_sr_lsp() but doesn't have the lsp endpoints because they are already set and we just need to update the path.

5.7.15 update_sr_lsp(odl_settings, name_of_lsp , pcc, sr_new_path)

Using above function this one update the LSP.

5.7.16 get_sid_ip_mappig(odl_settings)

Sometimes we may have a path just by IP addresses and may be only node SIDs.

This function use the topology information ODL gathered through the BGP-LS and returns two dictionary:

```
432    ''' This fuction returns below dictionaries'''
433    node_sid_to_ip = {} # only Node SIDs are unique, therefore we cannot map an Adj SID to an IP Address
434    ip_to_sid = {}        # We suppose that IP addresses are unique in the network and this program doesn't support subnet IP address for now
```

Node_sid_to_ip and **ip_to_sid** which are self-explanatory by their names.

```
436    url = 'http://{}:{}/restconf/operational/network-topology:network-topology/topology/{}'
437    url = url.format(odl_settings['odl_ip'], odl_settings['odl_port'], odl_settings['odl_bgpls_topology_id'])
438    headers={'content-type': 'application/json'}
439    result = get_url(url, headers, odl_settings)
```

Above piece of code gets the BGP-LS topology information from the ODL.

```
447    for node in nodes:
448        if "network-topology-sr:segments" in node.keys():
449            segments = node["network-topology-sr:segments"]
450            for seg in segments:
451                if "prefix" in seg.keys():
452                    if seg["prefix"]["node-sid"] == True:
453                        Nsid = seg["segment-id"]
454                        Nip = seg["prefix"]["prefix"].split('/')[0]
455                        SR_Nodes += 1
456                        # print(SR_Nodes, "N-SID : ", Nsid, "\tLoopback IP :", Nip)
457                        ip_to_sid.update({Nip : Nsid})
458                        node_sid_to_ip.update({Nsid : Nip})
```

Using above information, we check all nodes and extract the node SID and IP addresses from it.

5.7.17 covert_ip_path_to_sr_path(odl_settings, ip_path)

If we have an LSP consists of only IP addresses, this function using the above mentioned dictionaries, converts it to a proper SR-Path.

```
480  def covert_ip_path_to_sr_path(odl_settings, ip_path):
481      '''
482      SR-Path is a list where each item is a dict like below
483      {'sid': 10, 'ip-address': 1.1.1.1}
484      '''
485      dumm_dict, ip_to_sid = get_sid_ip_mappig(odl_settings)
486      sr_path = []
487      for ip in ip_path:
488          hop = {'ip-address' : ip, 'sid' : ip_to_sid[ip]}
489          sr_path.append(hop)
490      return sr_path
```

5.7.18 convert_nsid_path_to_sr_path(odl_settings, nsid_path)

This function do the same for a path consisting only SIDs.

```
508  def convert_nsid_path_to_sr_path(odl_settings, nsid_path):
509      ''' The nsid_path shall be only Node-SID hops '''
510      '''
511      SR-Path is a list where each item is a dict like below
512      {'sid': 10, 'ip-address': 1.1.1.1}
513      '''
514      node_sid_to_ip, dumm_dict = get_sid_ip_mappig(odl_settings)
515      sr_path = []
516      for nsid in nsid_path:
517          hop = {'ip-address' : node_sid_to_ip[nsid], 'sid' : nsid}
518          sr_path.append(hop)
519      return sr_path
```

5.7.19 Application APIs

URL: `/NodeSID_to_IP`

Method: `GET`

```
@app.route("/NodeSID_to_IP", methods=["GET"])
def get_NodeSID_to_IP():
    Dict_Nsid_ip, Dict_ip_sid = get_sid_ip_mappig(odl_settings)
    return jsonify({'NodeSID_to_IP':Dict_Nsid_ip})
```

URL: `/IP_to_SID`

Method: `GET`

```
@app.route("/IP_to_SID", methods=["GET"])
def get_IP_to_SID():
    Dict_Nsid_ip, Dict_ip_sid = get_sid_ip_mappig(odl_settings)
    return jsonify({'IP_to_SID':Dict_ip_sid})
```

URL: `/pcc_nodes`

Method: `GET`

```
@app.route("/pcc_nodes", methods=["GET"])
def get_pccnodes():
    pcc_nodes = get_pcc_nodes(odl_settings)
    return jsonify(pcc_nodes)
```

URL: `/pcc_nodes/node_ids`

Method: `GET`

```
@app.route("/pcc_nodes/node_ids", methods=["GET"])
def get_pcc_node_ids():
    pcc_nodes = get_pcc_nodes(odl_settings)
    pcc_node_ids = []
    for node in pcc_nodes:
        pcc_node_ids.append(node['node-id'])
    return jsonify(pcc_node_ids)
```

URL: `/pcc/<id>`

Method: `GET`

```
@app.route("/pcc/<id>", methods=["GET"])
def get_pcc_by_node_id(id):
    node_id = "pcc://" + str(id)
    node = get_pcc_node(odl_settings, node_id)
    return jsonify(node)
```

URL: `/pcc/<id>/reported_lsps`

Method: `GET`

```
@app.route("/pcc/<id>/reported_lsps", methods=["GET"])
def get_pcc_reported_lsps(id):
    node_id = "pcc://" + str(id)
    reported_lsps = get_reported_lsp(odl_settings, node_id)
    return jsonify(reported_lsps)
```

URL: `/pcc/<id>/reported_lsps/lsp_names`

Method: `GET`

```
@app.route("/pcc/<id>/reported_lsps/lsp_names", methods=["GET"])
def get_pcc_reported_lsps_names(id):
    node_id = "pcc://" + str(id)
    reported_lsps = get_reported_lsp(odl_settings, node_id)
    lsp_names = get_lsp_names(reported_lsps)
    return jsonify(lsp_names)
```

URL: `/pcc/<id>/reported_lsps/<lsp_name>`

Method: `GET`

```
@app.route("/pcc/<id>/reported_lsps/<lsp_name>", methods=["GET"])
def get_pcc_reported_lsp(id, lsp_name):
    node_id = "pcc://" + str(id)
    lsp = get_lsp(odl_settings, node_id, lsp_name)
    return jsonify(lsp)
```

URL: `/pcc/<id>/reported_lsps/<lsp_name>/path`

Method: `GET`

```
@app.route("/pcc/<id>/reported_lsps/<lsp_name>/path", methods=["GET"])
def get_pcc_reported_lsp_path(id, lsp_name):
    node_id = "pcc://" + str(id)
    lsp_path = get_lsp_path(odl_settings, node_id, lsp_name)
    return jsonify(lsp_path)
```

URL: `/pcc/<id>/reported_lsps/<lsp_name>/status`

Method: `GET`

```
@app.route("/pcc/<id>/reported_lsps/<lsp_name>/status", methods=["GET"])
def get_pcc_reported_lsp_status_all(id, lsp_name):
    node_id = "pcc://" + str(id)
    status_keys = [
                                    "processing-rule",
                                    "remove",
                                    "sync",
                                    "administrative",
                                    "ignore",
                                    "plsp-id",
                                    "operational",
                                    "delegate"
                            ]
    lsp_status ={}
    lsp = get_lsp(odl_settings, node_id, lsp_name)
    for key in status_keys:
         lsp_status.update({key : lsp["path"][0]["odl-pcep-ietf-stateful:lsp"][key]})

    return jsonify(lsp_status)
```

URL: `/add-lsp`

Method: `POST`

Content-Type: `json`

```
@app.route('/add-lsp', methods=['POST'])
def add_lsp():
    node_sid_to_ip, ip_to_sid = get_sid_ip_mappig(odl_settings)
    request_data = request.get_json()
    pcc = request_data['pcc']
    src = request_data['source-ipv4-address']
    dst = request_data['destination-ipv4-address']
    name_of_lsp = request_data['name']
    path = request_data['path']
    sr_path = []
    for hop in path:
        sr_hop ={ 'nai-type': 'ipv4-node-id'}
        if "sid" in hop.keys():
            sid = hop["sid"]
        else:
            sid = None
        if "ip-address" in hop.keys():
            ip = hop["ip-address"]
        else:
            ip = None
        # ============================================
        if ip:
            sr_hop.update({"ip-address": ip})
        else:
```

```
            if sid:
                if sid in node_sid_to_ip.keys():
                    ip = node_sid_to_ip[sid]
                else:
                    ip = None
                if ip:
                    sr_hop.update({"ip-address": ip})
                    sr_hop.update({"sid":sid})
                else:
                    return jsonify({"error":{"bad_hop": hop}})
            else:
                return jsonify({"error":{"bad_hop": hop}})
        # ============================================
        if sid:
            sr_hop.update({"sid":sid})
        else:
            if ip:
                if ip in ip_to_sid.keys():
                    sid = ip_to_sid[ip]
                else:
                    sid = None
                if sid!=None:
                    sr_hop.update({"sid":sid})
                    sr_hop.update({"ip-address": ip})
                else:
                    return jsonify({"error":{"bad_hop": hop}})
            else:
                return jsonify({"error":{"bad_hop": hop}})
        sr_path.append(sr_hop)

    if create_sr_lsp_ver2(odl_settings, src, dst, name_of_lsp , pcc, sr_path):
        return jsonify({})
    else:
        return jsonify({"error": "lsp creation faild!"})
```

5.8 Testing the Application

For testing our application, we build our code into a docker image and we used it to get the client commands through the APIs and perform the tasks. The container gets the requests and sends proper requests to the ODL subsequently.

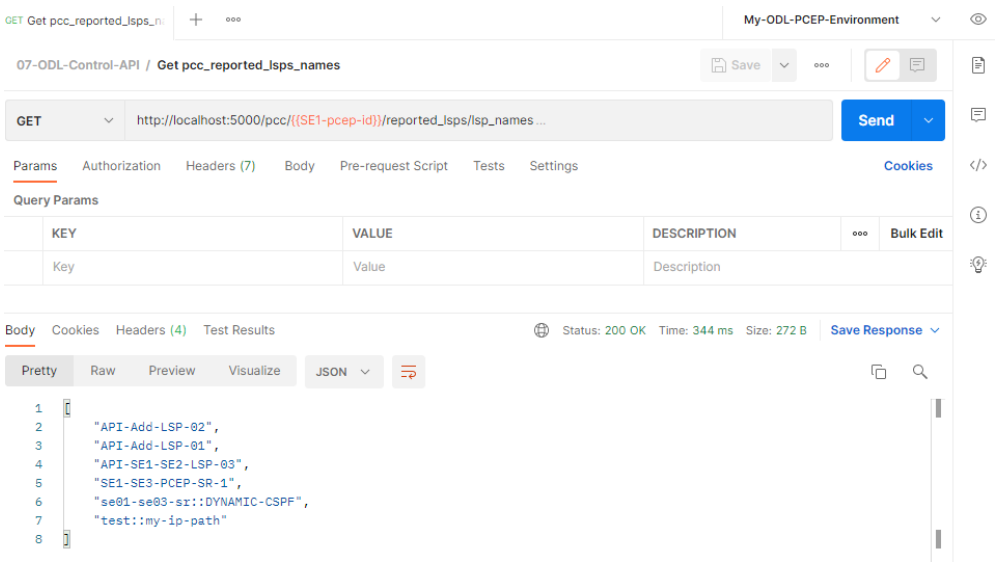
```
1  FROM python:slim-buster
2  RUN pip install flask --upgrade
3  RUN pip install requests
4  COPY odl_api_control.py .
5  CMD export FLASK_APP=odl_api_control.py && flask run --host 0.0.0.0
```

Figure 5-18 The dockerfile content

To send our request we use Postman.

```
PS F:\CapGitLab01\control-odl> docker run -p 5000:5000 -d flask-app
3fce5dfc2f3d395dcea3370ef2e706c1a93d7e4382b036a48f1bbbb95e7199f9
PS F:\CapGitLab01\control-odl>
```

Our image is named flask-app and it uses port 5000.



Above is the result of sending request to get the available LSPs on the SE1.

Now, we generate a new LSP named `API-SE1-SE2-LSP-04` from SE1 to SE2.

For our hops we use:

- LS1 SID
- GW1 IP Adr
- LS3 SID and IP Adr
- LS4 IP
- SE2 IP

We choose these hops to check if our function can find the counterpart IP or SID from the network topology and generate proper SR-Path. Below, you can see the json data we pass to our application using add-lsp api.

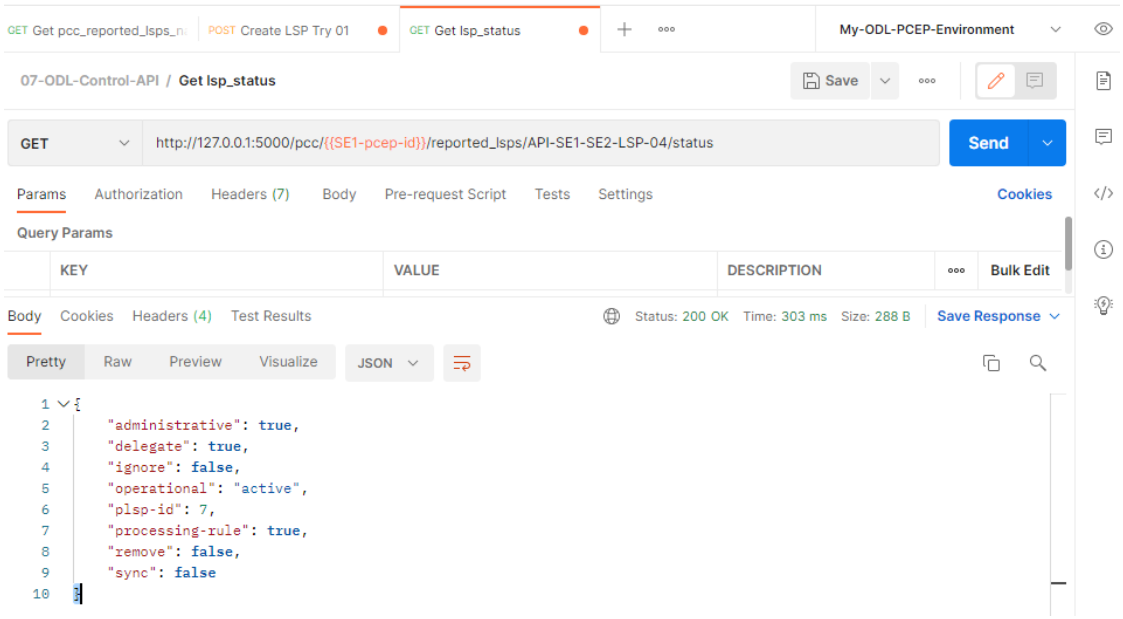
```
{
    "pcc": "pcc://{{SE1-pcep-id}}",
    "name": "API-SE1-SE2-LSP-04",
    "source-ipv4-address": "{{SE1-ip}}",
    "destination-ipv4-address": "{{SE2-ip}}",
    "path": [
                {
                    "sid": {{LS1-sid}}
                },
                {
                    "ip-address": "{{GW1-ip}}"
                },
                {
                    "sid": {{LS3-sid}},
```

```
                "ip-address": "{{LS3-ip}}"
            },
            {
                "sid": {{LS4-sid}},
                "ip-address": "{{LS4-ip}}"
            },
            {
                "ip-address": "{{LS2-ip}}"
            },
            {
                "ip-address": "{{SE2-ip}}"
            }
    ]
}
```

Now we use /pcc/{{SE1-pcep-id}}/reported_lsps/lsp_names to check if our LSP is generated.

```
[
    "se01-se03-sr::DYNAMIC-CSPF",
    "test::my-ip-path",
    "API-Add-LSP-02",
    "API-Add-LSP-01",
    "API-SE1-SE2-LSP-03",
    "API-SE1-SE2-LSP-04",
    "SE1-SE3-PCEP-SR-1"
]
```

Now, by using the /pcc/{{SE1-pcep-id}}/reported_lsps/API-SE1-SE2-LSP-04/status we get the status for this LSP:

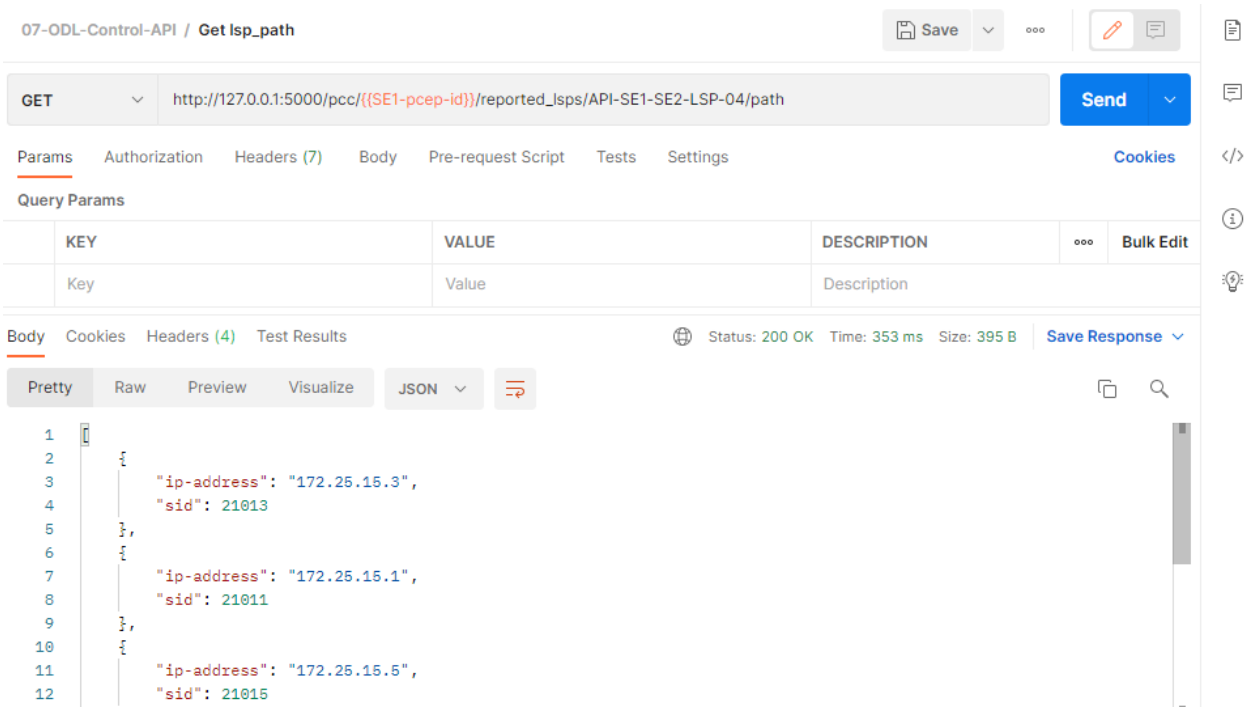


Our LSP is active and delegated to the PCE.

Now we update our LSP. Our path has hops as below:

- LS2
- LS1
- SE2

But before updating let’s get the current path.



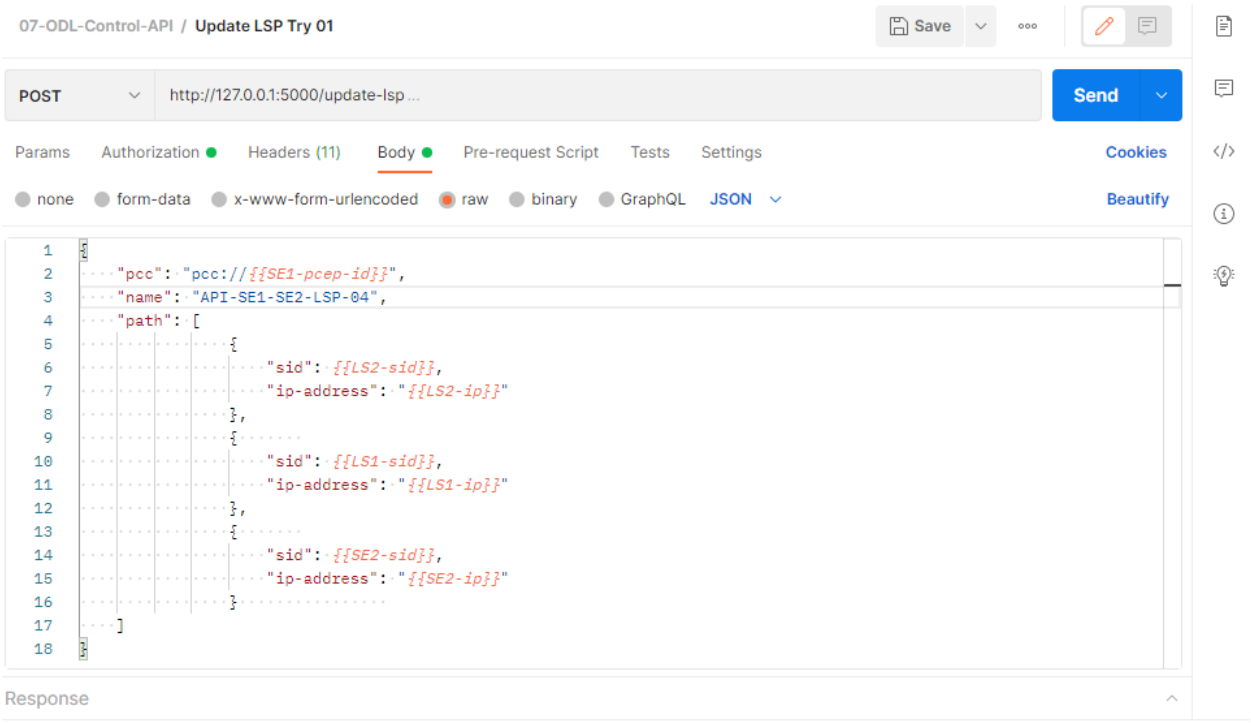
The complete path doesn’t fit in the above image. Below you can find the complete path:


```
[
    {
        "ip-address": "172.25.15.3",
        "sid": 21013
    },
    {
        "ip-address": "172.25.15.1",
        "sid": 21011
    },
    {
        "ip-address": "172.25.15.5",
        "sid": 21015
    },
    {
        "ip-address": "172.25.15.6",
        "sid": 21016
    },
    {
        "ip-address": "172.25.15.4",
        "sid": 21014
    },
    {
        "ip-address": "172.25.15.7",
```

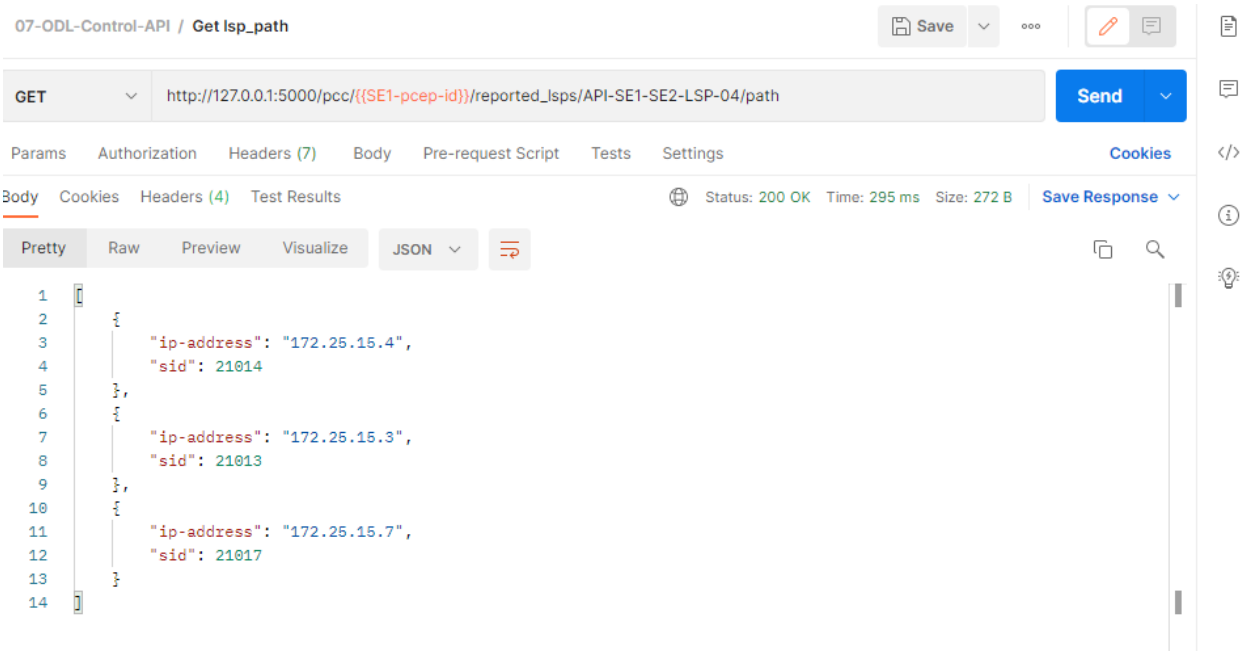
```
        "sid": 21017
    }
]
```

You can see that all hops have their SID and IP Adr.

New we update the LSP.



And we check the path again:



Which is what we passed to the application.

6 Conclusion

We started with a general overview of segment routing and its merits in the context of Software Defined Networking. The implementation included the creation of MPLS-TE and Segment Routing tunnels within the network utilizing the SDN controller, as well as the setup of the SDN controller and its connection to the network.

Then we discussed the PCEP protocol and how it can be utilized by a network controller to create, update and monitor SR-TE tunnels in the network. The controller we used, OpenDayLight, gives this advantage to the network administrator to have a complete network status and control by providing the link-state information gathered through the BGP-LS.

Opendaylight provides many APIs for the client to perform the required tasks. We wrote an application consisting of many functions to utilize these APIs and get the required information about our SR-TE LSPs, in addition to creating and updating them.

By adding our own APIs to the application, we added a higher layer that hides the ODL from the user. This application is what the user interface with. If there is another controller, the APIs for the user will remain the same, and just our code inside needs to be modified.

Adding another controller and providing adaptability to our application would be the next step.

7 References

[1] Ciena, "What is SDN?," Ciena, [Online]. Available: https://www.ciena.com/insights/what-is/What-Is-SDN.html. [Accessed 13 09 2021].

[2] C. Filsfils, S. Previdi, "SPRING Problem Statement and Requirements," *IETF draft-ietf-spring-problem-statement-07,* March 2016.

[3] S. Previdi, C. Filsfils, "Segment Routing with MPLS data plane," *IETF draft-ietf-spring-segment-routing-mpls-03,* February 2016.

[4] S. Previdi, A. Bashandy, C. Filsfils, "Segment Routing Architecture," *IETF draft-filsfils-rtgwg-segment-routing-00,* June 2013.

[5] S. Previdi, C. Filsfils, "Segment Routing Architecture," *IETF draft-ietfspring-*, December 2015.

[6] S. Previdi, C. Filsfils, "Segment Routing interoperability with LDP," *IETF draft-ietf-spring-segment-routing-ldp-interop,* October 2015.

[7] P. Francois, C. Filsfils, "Segment Routing Use Cases," *IETF draft-filsfilsrtgwg-segment-routing-use-cases-02,* October 2013.

[8] H. Gredler, P. Sarkar, "Anycast Segments in MPLS based Segment Routing," *IETF draft-psarkar-spring-mpls-anycast-segments-01,* October 2015.

[9] X. Xiao, "Traffic Engineering with MPLS in the Internet.," *Network, IEEE,* no. 14.2, pp. 28-33, 2000.

[10] [Online]. Available: http://www.mplsvpn.info/2015/07/segment-routing-based-mpls-vsclassic.html. [Accessed 8 Feb 2022].

[11] [Online]. Available: http://blog.ipspace.net/2011/11/ldp-igp-synchronization-in-mpls.html. [Accessed 8 Feb 2022].

[12] [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/mp_te_path_protect/configuration/xe-16-11/mp-te-path-protect-xe-16-11-book/mpls-traffic-engineering-fast-reroute-link-and-node-protection.html. [Accessed 8 Feb 2022].

[13] "Loop-Free Alternate (LFA) Applicability in Service Provider (SP) Networks," [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6571. [Accessed 4 Jan 2022].

[14] C. E. Hopps, "Analysis of an equal-cost multi-path algorithm," 2000.

[15] "Constrained Shortest Path First," [Online]. Available: https://www.metaswitch.com/knowledge-center/reference/constrained-shortest-path-first-cspf.

[16] "Segment Routing: Cutting Through the Hype," [Online]. Available: Segment Routing: Cutting Through the Hype and Finding the IETF's Innovative Nugget of Gold. [Accessed 5 Jan 2022].

[17] D. Singh, "PCE and PCEP Overview," [Online]. Available: https://packetpushers.net/pce-pcep-overview/. [Accessed 19 2 2022].

[18] J. P. Vasseur, J. L. Roux, "Path Computation Element (PCE) Communication," *IETF, RFC 5440,* Mar 2009.

[19] F. M. V. Ramos, E. Verissimo, D. Kreutz, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE,* vol. 103, no. 1, pp. 14-76, 2015.

[20] "What Is Software Defined Networking (SDN)? Definition," [Online]. Available: https://www.sdxcentral.com/resources/sdn/what-the-definition-ofsoftware-defined-networking-sdn/.

[21] S. Liu, "Segment Routing: Impact on Software Defined Networks," 27 March 2013. [Online]. Available: https://blogs.cisco.com/sp/segment-routing-impact-on-software-defined-networks.

[22] D. Jaksic, "Segment Routing in Service Provider networks," Cisco, Rovij, Croatia, 2018.

[23] "OpenDaylight Project, Donate Key Technologies to Accelerate Software-Defined Networking," [Online]. Available: https://www.linuxfoundation.org/press-release/industry-leaders-collaborate-on-opendaylight-project-donate-key-technologies-to-accelerate-software-defined-networking/.

[24] Z. K. Khattak, M. Awais, A. Iqbal, "Performance evaluation of OpenDaylight SDN controller," in *Parallel and Distributed Systems (ICPADS)*, 2014.

[25] "OpenDaylight, the Most Documented Controller," [Online]. Available: https://thenewstack.io/sdn-series-part-vi-opendaylight/.

[26] "PCE-Initiated LSPs," [Online]. Available: https://infocenter.nokia.com/public/7750SR217R1A/index.jsp?topic=%2Fcom.nokia.Segment_Routing_and_PCE_User_Guide_21.7.R1%2Fpcc-initiated_a-ai9ekdb667.html.

[27] "Getting Started Guide," Opendaylight, [Online]. Available: https://docs.opendaylight.org/en/stable-silicon/getting-started-guide/index.html.

[28] "PCEP LSP-DB API," [Online]. Available: https://docs.opendaylight.org/projects/bgpcep/en/latest/pcep/pcep-user-guide-active-stateful-pce.html#lsp-db-api.

[29] "LSP Deletion," [Online]. Available: https://docs.opendaylight.org/projects/bgpcep/en/latest/pcep/pcep-user-guide-active-stateful-pce.html?highlight=ipv4-adjacency#lsp-deletion.

[30] "LSP Deletion Using PCEP," [Online]. Available: https://infocenter.nokia.com/public/7750SR217R1A/index.jsp?topic=%2Fcom.nokia.Segment_Routing_and_PCE_User_Guide_21.7.R1%2Fpce-initiated_l-ai9ekdb7bn.html.

[31] "PCEP-ERROR Object Error Types and Values," [Online]. Available: https://www.iana.org/assignments/pcep/pcep.xhtml#pcep-error-object.

[32] R. Toghraee, Learning OpenDaylight, Birmingham: Packt Publishing Ltd, 2017.

[33] P. L. Ventre, S. Salsano, M. Polverini, "Segment Routing: A Comprehensive Survey of Research Activities, Standardization Efforts, and Implementation Results," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 1, pp. 182-221, 2021.

[34] F. Paolucci, F. Cugini, A. Giorgetti, N. Sambo and P. Castoldi, "A Survey on the Path Computation Element (PCE) Architecture," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1819-1841, 2013.

[35] Cisco, "Configure Segment Routing Path Computation," 18 8 2021. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/iosxr/ncs5xx/segment-routing/66x/b-segment-routing-cg-66x-ncs560.html. [Accessed 13 9 2021].

[36] "Segment Routing With Traffic Engineering (SR-TE)," Nokia, [Online]. Available: https://infocenter.nokia.com/public/7750SR217R1A/index.jsp?topic=%2Fcom.nokia.Segment_Routing_and_PCE_User_Guide_21.7.R1%2Fpcc-initiated_a-ai9ekdb667.html.